DESIGN AND IMPLEMENTATION OF A VIBRATION ANALYSIS TOOL

Design and Implementation of a Vibration Analysis Tool

By
Sahar Abughannam, B.Eng

A Thesis
Submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of

M.A.Sc
Department of Computing and Software
McMaster University

© Copyright by Sahar Abughannam, September 2008

MASTER OF APPLIED SCIENCE(2003) McMaster University
Hamilton, Ontario

TITLE: Design and Implementation of a Vibration Analysis Tool

AUTHOR: Sahar Abughannam, B.Eng(McMaster University)

SUPERVISOR: Dr. Martin von Mohrenschildt

NUMBER OF PAGES: xii, 123

Abstract

This thesis work presents the software development stages for a vibration analysis tool developed for a company that produces mining machinery and is interested in monitoring their machines' behaviour. The developed application contributes directly to the field of vibration analysis which is widely applied in many areas nowadays. Such areas include in addition to the mining industry, the automotive industry, and other industries producing rotating machinery.

The application developed interfaces with a device designed to acquire vibration data through a three axis acceleration sensor. The device then sends the data wirelessly via Bluetooth technology to a computer on which the application is running. The application then processes the data by applying signal conditioning functions on it. These functions include filtering, Fast and Fourier Transforms. Plots are then generated in order to monitor the machine's behaviour. Additionally, the unfiltered data is logged into a file which allows for playing it back for a more in depth analysis in the future. The application also provides a reporting system that generates reports providing a summary of the machine's behaviour and recommendations for adjusting the machine in order to obtain a better performance.

This application provides the company with a reliable and practical way of monitoring their machines, and acts as a solid base for future research that could involve diagnostics and control.

Acknowledgements

I would like to thank my supervisor Dr. von Mohrenschildt for his guidance and support over my graduate studies period. Also, the co-operation of the engineers at the company the software was developed for, as well as the company's financial support as part of an Industrial NSERC scholarship is greatly appreciated. Jay Parlar, who is a fellow graduate student working on a Ph.D. thesis related to this work, provided a large amount of suggestions and support throughout the implementation phase of the project, and I would like to acknowledge him for that. Finally, I would like to gratefully thank my husband Majd for his love and tremendous support, as well as my family members for their continuous encouragement.

Contents

Abstract **iii**

Acknowledgements **iv**

1 Introduction **1**

1.1 Thesis Motivation 1

1.2 Thesis Objective 3

1.3 Thesis Contributions 3

1.4 Thesis Overview 4

2 Vibration Analysis **5**

2.1 Overview 5

2.2 Terminology and Units 8

2.3 Filtering 11

2.3.1 Overview 11

2.3.2 The Butterworth Bandpass Filter 14

2.4 Fundamental Frequency Estimation 17

2.5 G Force, RPM, and Stroke Calculations 20

2.5.1 G Force 20

2.5.2 RPM 21

2.5.3 Stroke 21

2.6 Frequency Analysis 21

2.6.1 Fast Fourier Transform 22

2.6.2 Fast Wavelet Transform 23
2.7 Orbit Analysis 24
2.8 Waveform Analysis 26
2.9 Ellipse Fitting 28
2.9.1 Ellipse Fitting Method 29
2.9.2 Phase Angle 29
2.9.3 Eccentricity 29

3 Data Acquisition 31
3.1 Overview 31
3.2 Hardware Overview 32
3.3 PIC 33
3.4 Accelerometer 34
3.5 Bluetooth Device 34
3.6 Calibration 35

4 Requirements Specification 39
4.1 Overview 39
4.1.1 User Classes and Characteristics 40
4.1.2 User Documentation 41
4.1.3 Assumptions 41
4.2 Functional Requirements 42
4.2.1 Communication Requirements 42
4.2.2 Mode Requirements 42
4.2.3 Signal Processing Requirements 43
4.2.4 Logging Requirements 44
4.2.5 Display Requirements 44
4.2.6 Graphical User Interface Requirements 45
4.2.7 Reporting Requirements 46
4.3 Nonfunctional Requirements 48
4.3.1 Performance Requirements 48

4.3.2 Platform Requirements 49
4.3.3 Safety Requirements 49
4.3.4 Maintainability Requirements 49
4.3.5 Reliability Requirements 50
4.3.6 Usability Requirements 50

5 Software Design **52**
5.1 Overview 52
5.2 Design Method 52
5.3 Main Modules 55
5.3.1 Reader 55
5.3.2 Sensor Manager 57
5.3.3 Viewer 57
5.3.4 Logger 57
5.3.5 Calibrator 58
5.3.6 Filter 58
5.3.7 FFT Computer 58
5.3.8 Numerical Values Computer 59
5.3.9 Ellipse Fitter 59
5.3.10 Report Generator 59
5.4 Main Module Interaction 59

6 Software Implementation **61**
6.1 Overview 61
6.2 Implementation Considerations 61
6.3 Software Libraries Used 64
6.4 Modules 65
6.4.1 Reader 65
6.4.2 Sensor Manager 70
6.4.3 Viewer 72
6.4.4 Logger 76

6.4.5 Calibrator 76
6.4.6 Filter 77
6.4.7 FFT Computer 78
6.4.8 Numerical Values Computer 80
6.4.9 Ellipse Fitter 81
6.4.10 Report Generator 82
6.4.11 Device Discovery 83
6.4.12 Unpacker 83
6.4.13 Screen Scaling 84
6.4.14 Circular Queue 85

7 Software Testing **87**
7.1 Overview 87
7.2 Unit Testing 87
7.3 System Testing 88

8 Error Analysis **91**
8.1 Overview 91
8.2 Types of Errors 91

9 Conclusion **94**
9.1 Discussion 94
9.2 Future Work 95

A User Manual **99**
A.1 Start Up 99
A.2 File Simulation Mode 99
A.3 Data Acquisition Mode 103
A.4 Settings 107
A.5 Generating Reports 112
A.5.1 Other Commands 113

B Internationalization **116**

B.1 Editing the translation files 116

B.2 Adding a language 117

B.3 Adding new strings to the source code 117

C Reports **119**

C.1 Single Point Report 119

C.2 Orbit Summary Report 121

C.3 Tuning Report 123

List of Tables

3.1 Calibration measurements using gravity 36

8.1 G ranges and their resolutions 92

List of Figures

1.1 Types of machines produced and their motions. 2

2.1 Part of the interface of the developed Vibration Analysis Tool. 7
2.2 Measurement points on a two-bearing (horizontal) screen. 10
2.3 Measurement points on a four-bearing (inclined) screen. 10
2.4 Representation of a bandpass filter 13
2.5 Frequency response of Butterworth filters of varying order. 15
2.6 Poles of a 4th order Butterworth filter. 16
2.7 DFT example with an integer frequency value. 19
2.8 DFT example with a non integer frequency value. 19
2.9 FFT example 23
2.10 Filtered vs. Unfiltered Orbit Example 25
2.11 Filtered Orbits and their FFTs. 26
2.12 Waveforms generated by a vibrating machine in the X,Y and Z axes . 27
2.13 Waveforms generated by a vibrating machine with noise. 28
2.14 Ellipse example showing major and minor axes. 30
2.15 Examples of orbital trajectories with various eccentricities 30

3.1 Portable machinery vibration analyser and data logger. 32
3.2 Data Acquisition Hardware Prototype. 33
3.3 Best fit line for x axis measurements under 2.5 g mode. 37

4.1 A waterfall process with feedback. 40

5.1 Data Flow in the System. 53
5.2 The model-view-controller architecture. 54
5.3 Alternate model-view-controller architecture. 55
5.4 Depiction of the main system loops. 56
5.5 Module interaction diagram. 60

6.1 An illustration of the circular queue data structure. 86

A.1 Language Selection Menu. 100
A.2 Mode Selection Menu. 101
A.3 Main view of File Simulation Mode. 102
A.4 Main view of File Simulation Mode while running. 104
A.5 Acquisition Information Gathering Menu. 105
A.6 Device Selection Menu 106
A.7 Main view of Data Acquisition Mode. 108
A.8 Centre frequency tolerance setting. 109
A.9 RPM tolerance setting. 109
A.10 Update frequency setting. 109
A.11 Update history setting. 110
A.12 FFT size setting. 111
A.13 Acceptance levels setting. 111
A.14 Bearings settings. 112
A.15 Tuning Report Menu 114

C.1 Example of a single point report. 120
C.2 Example of an orbit summary report. 122
C.3 Example of a tuning report. 124

Chapter 1

Introduction

1.1 Thesis Motivation

The field of modern vibration analysis (also referred to as condition monitoring) for rotating machinery is now over 40 years into its development and thus is a matured technical subject. However, it continues to evolve and advance in response to new requirements to further reduce machinery downtime and drastically reduce maintenance costs [1]. This thesis work is part of this advancement.

We were approached by a company that produces vibrating screen machinery which is used mainly in the field of mining. Figure 1.1 illustrates a portion of the types of machines produced, along with their resulting motions. Two main motions are produced; circular and horiztonal based on the application the machine is used in. As the cut size of the vibrating screen increases, its amplitude increases as well. Low cut size - low amplitude machines are typically used for dry screening of fine materials such as lime, chemicals, fertilizers, sand, and have very high frequencies. Whereas, high cut size - high amplitude machines are used for course materials such as coal and phosphate rock and can rotate at speeds as high as 1050 RPM.

The company had an existing vibration analysis tool which they wished to improve. The existing setup was such that only one part of the vibrating machine could be monitored at once. The three axis accelerometer which would acquire the vibra-

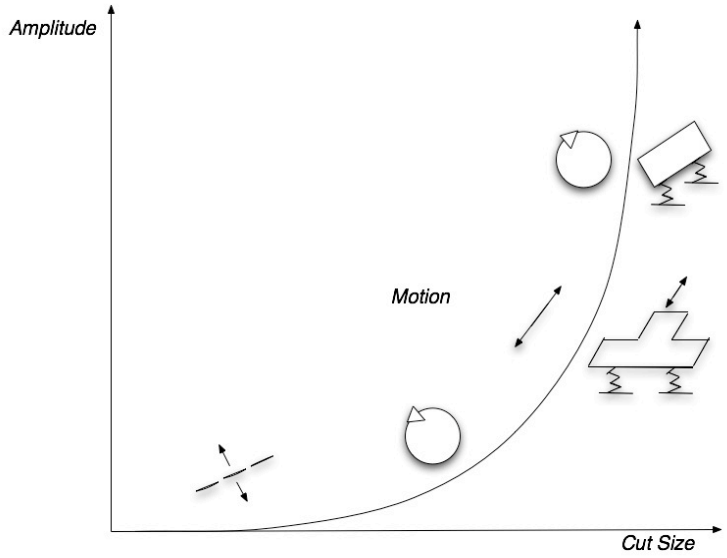


Figure 1.1: Types of machines produced and their motions.

tion data from one point on the vibrating screen was connected through a wired serial connection to a laptop which had the vibration analysis tool on it. The company's desire to replace this setup was for two reasons. First and foremost, for the safety of its technicians who would have to carry a laptop attached via a wire to the vibrating screen. Secondly, acquiring data from different points on the vibrating screen at different times did not prove to be useful. The purpose of vibration monitoring is to gain as much information about the screen *at once*, and that was not the case. Other reasons existed for the company's desire to replace their current solution, one of which is in the method of computing the centre frequency used in the bandpass filter that was utilized in filtering the acquired data. The current method was based on finding the maxima points on a signal and calculating the period of time between them. The period was then inverted to calculate the frequency of the signal, which was used as the centre frequency of the bandpass filter. This method did not give reliable results for noisy signals, and a better method was required. Additionally, the current hardware was very expensive to produce, and they wished to replace it with a hardware that is as efficient, yet with a less expensive price. Finally, the company was open to

any new ideas that could be added to the current vibration analysis tool which could improve its efficiency and possibly allow for additional features. This thesis work is mainly a development of a vibration analysis tool that fits this company's needs.

1.2 Thesis Objective

The goal of this thesis work is the development of a complete vibration analysis application that performs at least the following:

- Obtain readings from the acquisition device introduced in Chapter 3.
- Perform signal conditioning on the obtained readings. This includes, but is not limited to, applying filters on the data, performing calculations to obtain numerical values such as RPM, stroke and acceleration, and running an FFT (Fast Fourier Transform) on the data.
- Produce plots for each of the monitored locations representing the 2-dimensional motion acquired from a specific location, the waveforms representing the signals coming from each of the individual axes, and the frequency spectrum produced from applying an FFT on the data.
- Log the data and allow for playing it back in the future.
- Generate three types of reports: Single point (for each of the locations monitored), Orbit report (a summary of all monitored locations), and Tuning report (providing recommendations to tune and adjust the machine).

1.3 Thesis Contributions

The main contribution of this thesis work is in developing a vibration analysis tool that can be used in wirelessly monitoring multiple locations on a vibrating machine at once, and producing reports that provide recommendations for adjusting the machine. The

developed tool allows researchers and engineers to conduct work in order to diagnose certain machine faults, and to ultimately control the machine's behaviour.

1.4 Thesis Overview

This thesis is divided into the following chapters reflecting when appropriate the software design life-cycle:

Chapter 2 introduces the field of vibration analysis, and discusses the various methods that are practised when applying it.

Chapter 3 discusses the hardware used for acquiring data prior to performing the analysis on it, as well as an explanation of the method used for calibrating the sensors used for the acquisition.

Chapter 4 acts as a requirements document for the developed vibration analysis system, discussing both the functional and the non-functional requirements of the system.

Chapter 5 serves as a software design document explaining the design decisions made as well as the design architecture used. It also lists the software modules that the application consists of, along with a description of how they interact.

Chapter 6 describes the implementation of the vibration analysis tool, including the classes constructed in the implementation phase and how each one of these classes is implemented.

Chapter 7 discusses the testing stage of the developed tool.

Chapter 8 is a discussion of errors that affect the data and their tolerances.

Chapter 9 is a summary of the thesis work, along with some suggested future work.

Appendix A is a user manual explaining how to use the application.

Appendix B explains the method of internationalization and the steps that need to be taken in order to apply it on a new language.

Appendix C contains the three different types of reports generated by the application.

Chapter 2

Vibration Analysis

2.1 Overview

Vibration is the most regularly measured condition parameter in modern rotating machinery, and is nowadays continuously monitored in numerous important applications [1]. Such applications appear in various energy and process industries such as petrochemical, refining, power generation and numerous others [14]. Vibration monitoring (or analysis) was first used in the mid 1950s [1] when vibration sensors were first developed. At that time, the main use of vibration monitoring was in providing warnings when a machine was producing excessively high frequency vibrations that could potentially damage its parts. Numerous experimental results have been reported showing that high frequency vibrations are associated with bearing deterioration [8]. This application of vibration monitoring is refered to as *preventive maintenance*.

Nowadays, vibration monitoring is being used more extensively. *Predictive maintenance*, which is an extension of the traditional *preventive maintenance*, is one of the most common applications. One version of predictive maintenance is where the machine is provided specific maintenance actions based upon its monitored condition instead of a fixed-time maintenance cycle [1]. The motivation behind predictive maintenance is in reducing maintenance costs. This cost reduction is mainly done

by making large reductions in the amount of maintenance required, in addition to reducing technical support personnel.

Many methods are used in performing vibration analysis. As discussed in [1], such methods include but are not limited to the following:

- **Signal Conditioning**: This includes *filtering* (discussed in Section 2.3, *integration* to calculate the velocity and displacement from the acceleration (discussed in Section 2.5), and *signal amplitude conversion* which is part of the Fourier analysis (discussed in Section 2.6).
- **FFT Spectrum**: This is discussed in Section 2.6.
- **Rotor Orbit Trajectories** : This is discussed in Section 2.7
- **Bode, Polar and Spectrum Cascade Plots** : Not a part of this thesis work.
- **Wavelet Analysis Tools** : This is partially discussed in Section 2.8
- **Chaos Analysis Tools** : This is not a part of this thesis work.

In addition to the above, this thesis work involves not only visual orbit analysis, but a mathematical analysis including the fitting of the orbit trajectory into an ellipse and the calculation of the ellipse's phase and eccentricity.

Figure 2.1 provides a layout of a section of the Vibration Analysis Tool developed as part of this thesis work. Part **A** displays the Fast Fourier Transforms for the three axes. Part **B** displays the waveforms for the three axes. Part **C** is a depiction of the orbit in the Y vs. X orientation. The orbit in this figure is based on the filtered data, however there is an option to view the orbit based on the unfiltered data in the options panel (to the right of Part **C**). Part **D** is a smaller representation of the orbit for a specific sensor assigned to a specific location. Part **E** is the *values panel* where the *acceleration*, *stroke*, *RPM*, *phase* and *eccentricity* values are displayed.

It is important to understand that the values read from the sensors are accelerations in the X,Y and Z directions. These values include three different types of data:

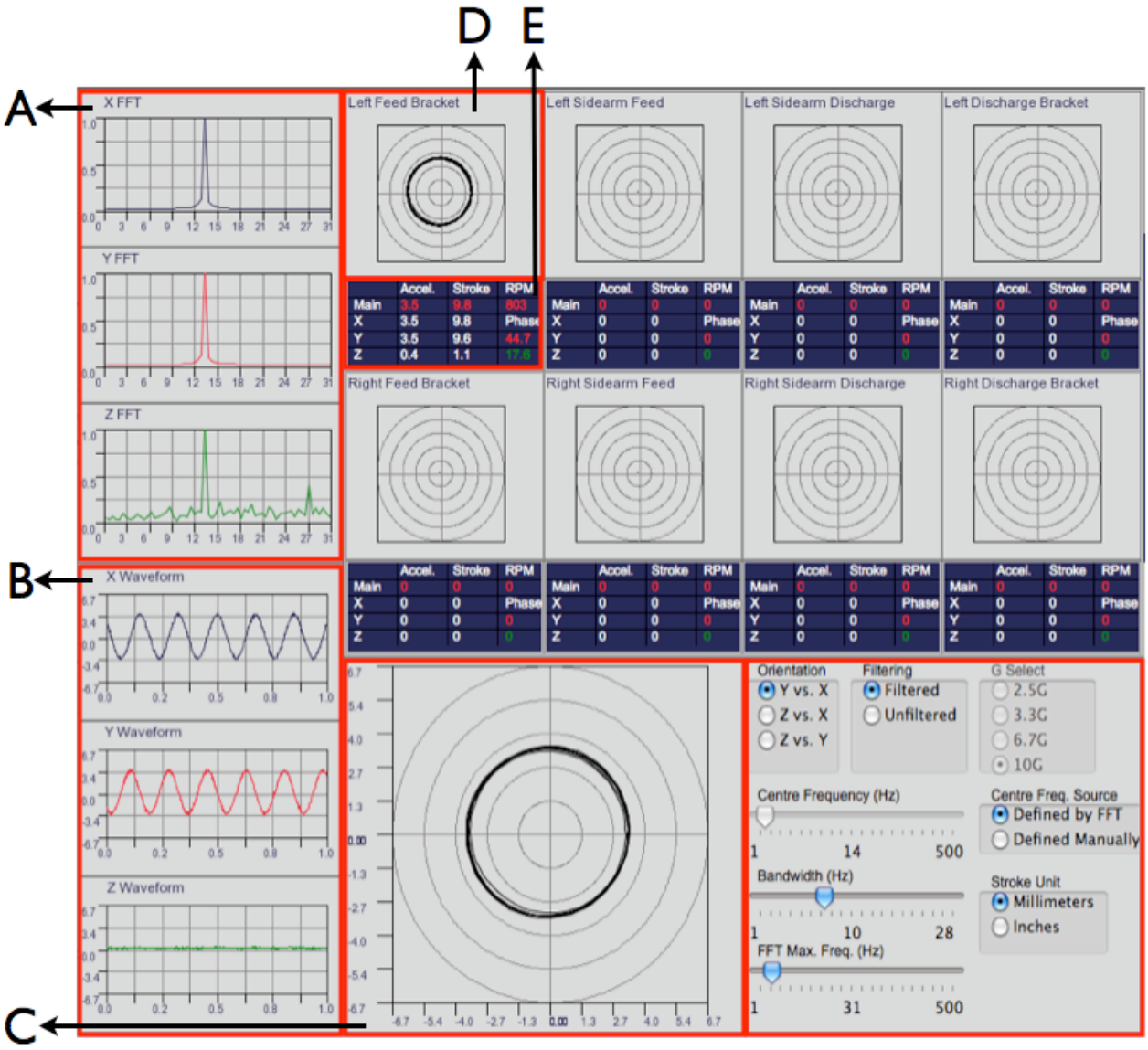


Figure 2.1: Part of the interface of the developed Vibration Analysis Tool.

1. *Wanted* data representing the normal motion of the machine that it was designed for.

2. *Unwanted* data representing the abnormal motion of the machine produced from a mechanical problem.

3. *Noise* coming from other sources such as the sensor units or the surrounding environment.

Based on the discussion above, the vibration analysis tool will have to process the data in such a way that the wanted data is kept, the unwanted data is reduced and the noise is completely eliminated.

This chapter first introduces some terminology commonly used in the field of vibration analysis, and then discusses a few methods that are used in this thesis work to perform the analysis.

2.2 Terminology and Units

The following terms are frequently used in the field of vibration analysis and are worth defining here:

- **RPM**: The number of revolutions the machine performs in a minute. This is equivalent to $60 \times f_0$ where f_0 is the fundamental frequency of the machine.

- **Stroke**: The radius of the circle that the machine's main movement creates. This quantity is measured in mm or $inches$.

- **g Force**: The acceleration of the machine in a certain direction. This quantity is measured in g which is equivalent to $9.8m/s^2$.

- **Numerical Values**: The g force, RPM, stroke, phase and eccentricity (refer to Section 2.9) values based on the data coming from a sensor mounted to a specific location.

- **Running Frequency**: The main frequency of the machine (also referred to as the *Fundamental Frequency* (f_0) which represents the main movement that the machine was designed to have. This quantity is measured in Hz.

- **Bandwidth**: The range of frequencies in the bandpass filter, where all the frequencies in this range are passed through the filter, and frequencies outside this range are removed. This quantity is measured in Hz.

- **Sampling Rate**: The rate at which the data read by the sensor is sampled. The sampling rate is measured in Hz.

- **Material Flow**: This is the direction that the material moves. It starts from the feed end of the machine and ends at the discharge end.

- **Sensor Mounting**: When mounting a sensor on the vibrating screen, the orientation should be in a such a way that the x-axis of the sensor is aligned with the screen's inclination and the positive side of the x-axis points toward the screen's feed end. If a magnet is used to mount the sensor, then the magnet should sit tight on the surface of the machine.

- **Sensor Locations**: The predefined locations that the sensors are mounted to on the vibrating screen and named as follows:

- **LFB (RFB)**: Left (Right) Feed-end bracket.

- **LDB (RDB)**: Left (Right) Discharge-end bracket.

- **LSF (RSF)**: Left (Right) Sidearm Feed-end.

- **LSD (RSD)**: Left (Right) Sidearm Discharge-end. On the two-bearing screen machines, the positions are marked as in Figure 2.2, whereas on the four-bearing screen machines, the positions are marked as in Figure 2.3.

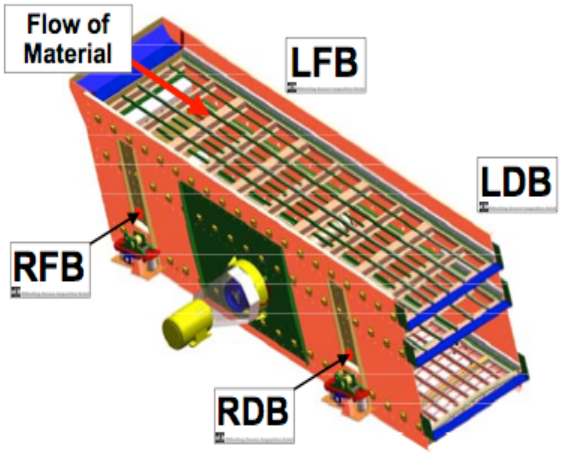


Figure 2.2: Measurement points on a two-bearing (horizontal) screen.

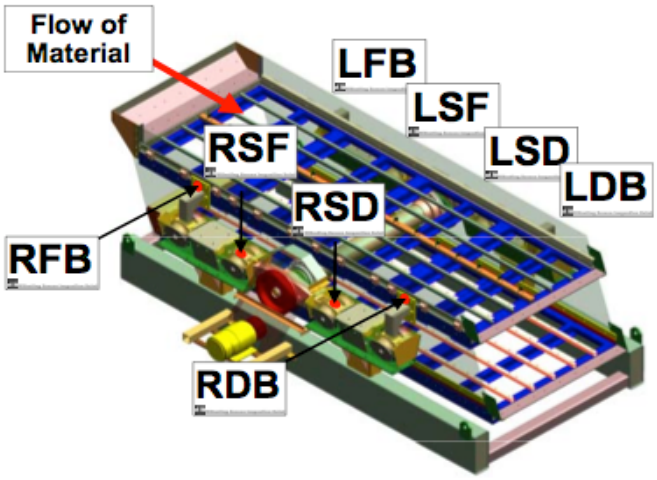


Figure 2.3: Measurement points on a four-bearing (inclined) screen.

2.3 Filtering

2.3.1 Overview

Raw vibration signals coming directly from the acquiring sensor, always contain components which are contamination (noise), and frequently some actual components which may partially obscure other components that are the important part of the measurements taken [1]. This makes *filtering* the most common signal conditioning operation in such applications.

Prior to filtering the digital data once acquired and before passing it through the Analog to Digital (AD) convertor, low-pass analog filtering is inserted ahead of the AD converter to avoid *aliasing*. Aliasing causes the "reflection" into the lower end of the spectrum of high-frequency content that is above the sampling-rate capability of the AD convertor [1]. Therefore, the low-pass analog filter should have a cutoff frequency sufficiently below the Nyquist frequency, which is $\frac{1}{2}$ the sampling frequency. The reason that the cutoff frequency of the analog filter has to be *substantially* below the Nyquist frequency, is because no analog filter has a perfect frequency cutoff, i.e. it has its rolloff *above* the cutoff frequency [1].

On the digital side, filters that are used commonly with vibration signals are *low-pass*, *high-pass*, *band-pass*, *notch* and *tracking* filters. The following is a brief description of each of these types.

- *Low-pass* filters are the most signal conditioning tool used in handling vibrating machinery signals [1]. A low-pass filter works by attenuating signal content above a specified frequency (the *cut-off* frequency) and passing the signal content below it. Since for routine rotating machinery vibration analysis, frequency components above ten times the spin speed are usually not of interest [1], the low-pass filter meets the requirement.

- *High-pass* filters work in a way opposite of the low-pass filters. Signal content below the cutoff frequency is attenuated, and that above the cut-off frequency is passed. As mentioned above, since routine rotating machinery vibration

analysis is usually not concerned with frequency components above 10 times the spin speed, high-pass filters are not a common tool in rotating machinery signal conditioning.

- *Band-pass* filters operate based on a range of frequencies called the *frequency band* of the filter. A band-pass filter's operation is similar to applying a low-pass filter and a high-pass filter in *series*, with the condition that the low-pass filter's cut-off frequency is *higher* than the high-pass filter's cut-off frequency. The frequency band of the band-pass filter is centred at a point called the *centre frequency* of the filter. This frequency is determined first, and then a range is defined around it allowing signal content within that range to be passed, and that outside it to be removed. The standard method of determining the centre frequency of the band-pass filter in rotating machinery applications is by using the machine's speed, also referred to as its *RPM*. For purposes of rotor balancing, only the synchronous vibration components are used [1]. The reason for this is that the application of balancing is based on the assumption that the vibrating system overall is linear. This means that only the forcing frequency (which occurs once per revolution) vibration amplitude and phase angle are accommodated in the balancing procedures [1]. In general, synchronous band-pass filtering increases the accuracy of rotor balancing which is an important procedure in the field of rotating machinery.

  Figure 2.4 is an amplitude-vs-frequency representation of a bandpass filter. The frequencies f_1 and f_2 are the cut-off frequencies of the filter, and f_0 is the centre frequency. The cutoff frequencies are those at which the output signal power falls to half of its level at f_0. The difference between f_1 and f_2 is referred to as the *bandwidth* of the filter. Finally, the range of frequencies between f_1 and f_2 is referred to as the *filter passband*.

- *Notch* filters operate in a such a way opposite to band-pass filters. Instead of passing the signal content within a specified range, the content outside that range is passed and that within it is removed. One interesting application

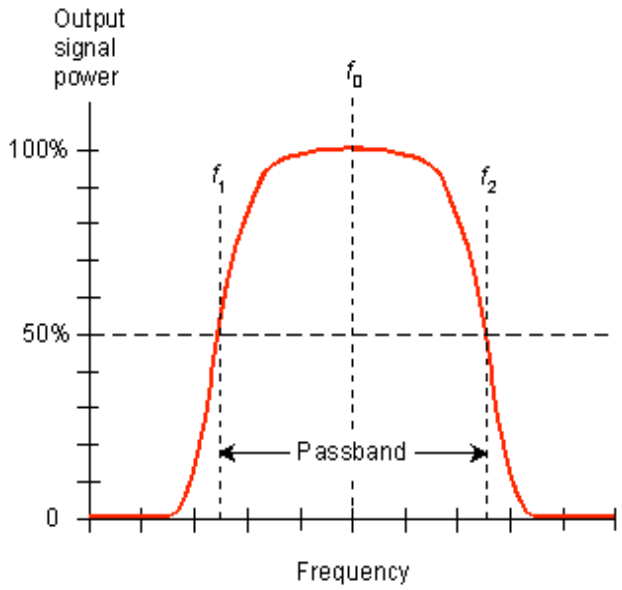


Figure 2.4: Representation of a bandpass filter.

of notch filters in rotating machinery is with *magnetic bearings*. Since the operation of magnetic bearings is based on displacement feedback control, a notch filter us used to filter out the bearing force components occurring once per revolution, so that they are not transmitted to the non-rotating structure of the machine, while the bearings continue to provide static load support capacity and damping [1].

- *Tracking* filters are any of the above filters with the added feature that the cutoff frequency (or frequencies in the case of the band-pass and notch filters), is made to track a specified signal component. In rotor vibration analysis, the main application of tracking filters is when using a band-pass filter and tracking its centre frequency. As the machine is slowly brought up to its operating speed or brought down to shutoff, its RPM is changing and is therefore tracked and used as the filter's centre frequency.

Based on the previous discussion, it is clear that the most effective filter in terms of rotating machinery signal processing is the band-pass filter. The next section

discusses a specific type of band-pass filter, namely the *Butterworth* filter, which is used as part of this thesis work.

2.3.2 The Butterworth Bandpass Filter

The most commonly used method for the design of IIR[1] *digital* filters is the bilinear transformation of the classical *analog* filters [10], such as Chebyshev Type I and Type II, the Bessel filter and the Butterworth filter. This section will discuss the design of a digital IIR Butterworth filter.

Introduced by Butterworth (1930), the Butterworth filter is one of the earliest systematic analog filter design methods. It is also one of the most widely used due to its *maximally flat* property. The main feature of the Butterworth filter is that its frequency response is maximally flat in the passband, and rolls off towards zero in the stopband. This allows frequencies to be passed without distortion, or causes them to be removed completely. Figure 2.5 (taken from [12]) displays the frequency response of a number of Butterworth filters with orders from 0 to 10. It can be seen that as the order N increases, the filter approaches an ideal low pass filter.

When designing a Butterworth bandpass filter, two quantities have to be determined. First, the filter's order, and second its cutoff frequencies (based on its centre frequency and bandwidth).

First, since the current vibration analysis tool, that this thesis work is an extension for, uses a 4th order Butterworth filter, it was decided to use the same order. Higher order filters would provide a "cleaner" filtering, however, they require additional computations in order to calculate their coefficients. Since such an application is performed real-time, and speed is a major factor, there is a tradeoff when selecting the order of the filter. However, there exists a mathematical method for determining the *minimum* required order. Let the filter's *passband* be between w_p (frequencies greater than this are *passed*) and w_s (frequencies less than this are *stopped*). Also,

[1] A filter with the property of Infinite Impulse Response. This means that the filter has an impulse response function which is non-zero over an infinite length of time, as opposed to the Finite Impulse Response filters whose impulse response is on a fixed duration.

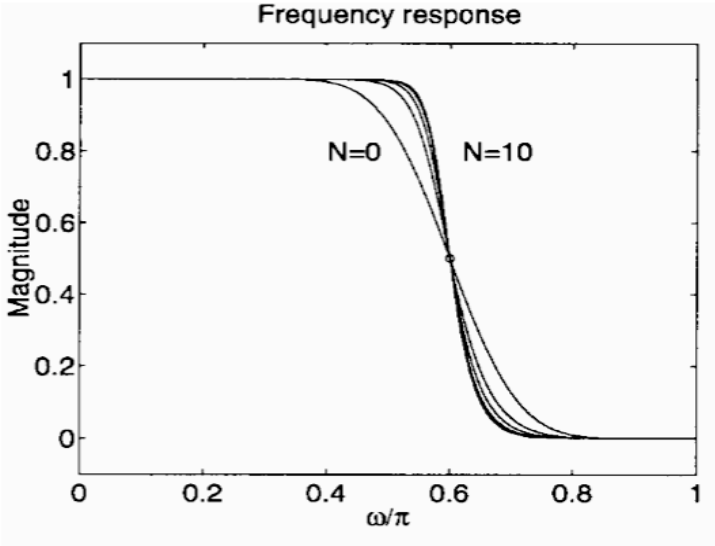


Figure 2.5: Frequency response of Butterworth filters of varying order.

let A_p be the permitted magnitude deviation within the passband, and A_s be the minimum attenuation relative to the passband peak response in the stopband. Both A_p and A_s are measured in dB. Then, the minimum Butterworth filter order to meet those specifications is given by (2.1).

$$N = \lfloor \frac{log[(10^{A_s/10} - 1)/(10^{A_p/10} - 1)]^{1/2}}{log(w_s/w_p)} \rfloor \tag{2.1}$$

Second, the centre frequency of the filter is to be determined. Section 2.4 explains exactly how this is done.

Once the *order* (N) and *centre frequency* (w_c) of the Butterworth bandpass filter are determined, the filter can be designed.

The general magnitude-squared frequency response for a transfer function with finite poles and all zeros at infinity is given as follows:

$$|Hjw|^2 = \frac{K^2}{\sum_{k=0}^{N} \alpha_k w^{2k}} \tag{2.2}$$

The coefficients K, α_k are to be determined. (2.2) can be expressed as follows:

$$|Hjw|^2 = \frac{K^2}{\alpha_0} \times \frac{1}{\sum_{k=0}^{N} \beta_k w^{2k}}, \tag{2.3}$$

where $\beta_k = \alpha_k/\alpha_0$. Performing long division and expansion as described in Chapter 3 of [3], the following equation is derived:

$$|Hjw|^2 = \frac{K^2}{\alpha_0} \times \frac{1}{1 + \beta_N w^{2k}}, \tag{2.4}$$

Finally, letting K^2/α_0 be unity and β_N be $1/w_c^{2N}$, the Butterworth response is defined as follows:

$$|Hjw|^2 = \frac{1}{1 + (w/w_c)^{2N}} \tag{2.5}$$

From 2.5 the Laplace Transform can be calculated as follows:

$$Y(s) = H(s)H(-s) = \frac{1}{1 + (s/jw_c)^{2N}} \tag{2.6}$$

Next, the poles of 2.6 are found by setting $(s_k/jw_c)^{2N} = -1$ and solving for the values of s_k. For a 4th order Butterworth filter normalized for $w_c = 1$, the 4 poles of $H(s)$ are: $e^{\pm j5\pi/8}$, $e^{\pm j7\pi/8}$. Figure 2.6 shows a plot of the poles of $H(s)H(-s)$. Note that the poles are equally spaced on the unit circle, and those corresponding to $H(s)$ appear on the left half of the s-plane.

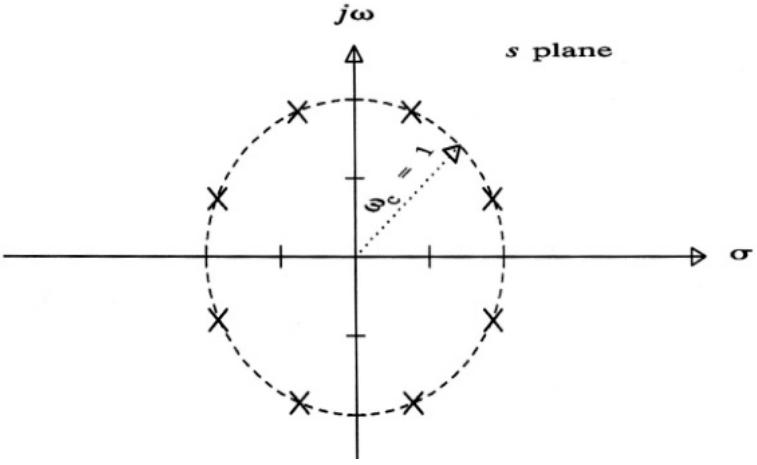


Figure 2.6: Poles of a 4th order Butterworth filter.

Finally, the transfer function $H(s)$ can be written as:

$$H(s) = \frac{1}{(s^2 + 0.7653669s + 1)(s^2 + 1.847759s + 1)} \tag{2.7}$$

Or equivalently,

$$H(s) = \frac{1}{s^4 + 2.613126s^3 + 3.414213s^2 + 2.613126s + 1} \tag{2.8}$$

The denominator of 2.8 is the fourth-order Butterworth polynomial. Comparing it to $H(s) = \frac{1}{\sum_{i=0}^{N} a_i s^i}$ the a coefficients of the denominator can be determined. The numerator coefficients are all zeros except for $b_0 = 1$.

Using the a and b coefficients determined above, a recursive IIR filter can be implemented as follows:

$$y(n) = \sum_{m=0}^{L-1} x(n-m)b(m) + \sum_{m=1}^{M-1} y(n-m)a(m) \tag{2.9}$$

Finally, the Butterworth filter is updated whenever the centre frequency or bandwidth change. Every time such an update is requested, the coefficients of the filter are recalculated.

2.4 Fundamental Frequency Estimation

Fundamental Frequency estimation is a popular topic in many fields of research. The need for detecting the fundamental frequency in vibration signal monitoring, is in the calculation of the rotating machine's RPM, as well as in determining the centre frequency for the utilized bandpass filter. Many techniques exist for estimating the fundamental frequency of a signal, but not all perform well when there exists noise in the signal. In [6], various methods are suggested for pitch detection (which is the same as fundamental frequency detection), both in the Time-Domain and the Frequency-Domain. Since in this thesis work, the FFT is calculated in order to perform the FFT analysis as described in 2.6, in the interest of speed, the calculated FFT can be used to estimate the fundamental frequency.

The FFT is an implementation of the DFT (Discrete Fourier Transform) which is one of the two most common and powerful procedures encountered in the field of digital signal processing (Digital filtering is the other) [9]. Let f_s be the sampling

frequency that the data is sampled at, and N be the number of samples that go through the DFT. Then, the DFT equation is given by:

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N} \tag{2.10}$$

Or in rectangular form:

$$X(m) = \sum_{n=0}^{N-1} x(n)[cos(2\pi nm/N) - jsin(2\pi nm/N)] \tag{2.11}$$

This shows that each DFT output term is the sum of the term-by-term products of an input time-domain sequence with sequences representing a sine and a cosine wave. The magnitude of the DFT results is directly proportional to N, and the DFT's *frequency resolution* is f_s/N. Note that as N increases, the DFT resolution becomes smaller, but the DFT evaluation becomes more expensive. Figure 2.7 which is taken from [9] is an example of a DFT of a signal whose frequency is an integer value equal to $3Hz$.

The fundamental frequency that is being detected in the case of rotating machinery, is the frequency with the highest magnitude component in $X(m)$. However, due to what is referred to as the *DFT leakage* problem, this is not the most accurate value. The *DFT leakage* problem arises when the input signal's frequency is not exactly at a DFT bin centre. This causes this frequency to *leak* into all of the other DFT output bins (see Figure 2.8 which is taken from [9]).

One solution to this problem is by performing a parabolic interpolation around the frequency with the highest magnitude. The following outlines the procedure for doing this:

1. The DFT is computed on the N samples.

2. The frequency with the highest magnitude in the DFT is determined. Let this frequency be the ith sample.

3. A parabolic interpolation on three points $(i-1, i, i+1)$ is performed to obtain an equation of a parabola $y(x) = ax^2 + bx + c$. The fundamental frequency has

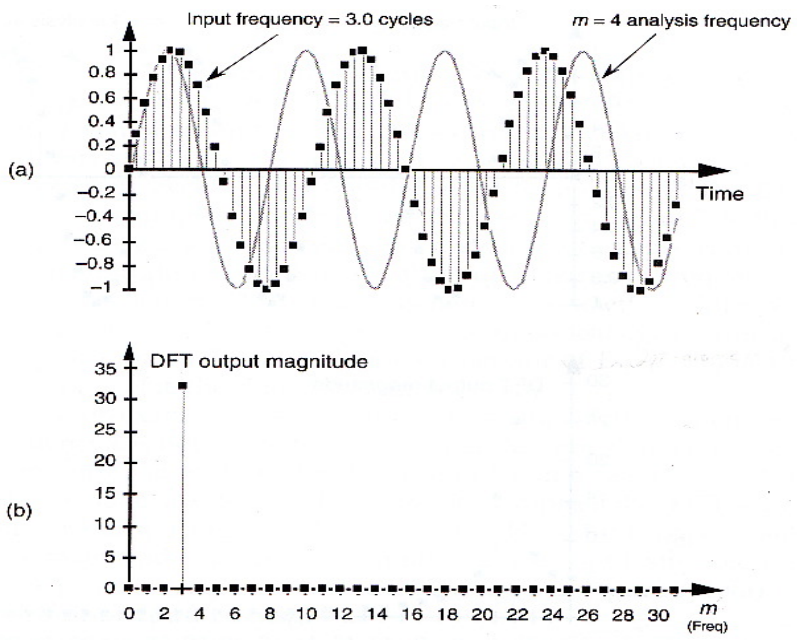


Figure 2.7: DFT on 64 samples: (a) the input signal with frequency equal to $3Hz$. (b) DFT output magnitude.

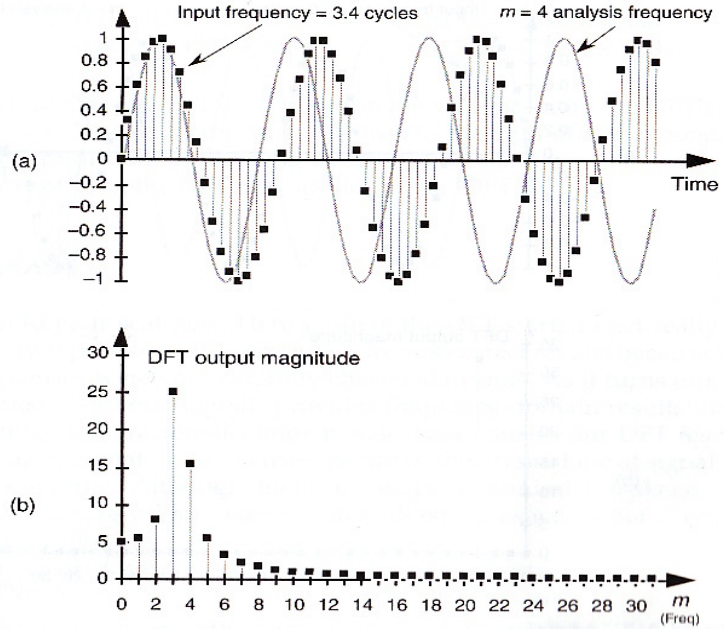


Figure 2.8: DFT on 64 samples: (a) the input signal with frequency equal to $3.4Hz$. (b) DFT output magnitude.

the highest magnitude, meaning it appears at the peak of the parabola where the slope is *zero*. Therefore, a better estimation of the fundamental frequency is computed by solving for the point where $2ax + b = 0$, where a and b are defined in the parabola's equation.

4. The result obtained from step (3) is multipled by the DFT *frequency resolution* to obtain the final estimate of the fundamental frequency.

2.5 G Force, RPM, and Stroke Calculations

Other than filtering, signal conditioning approaches include *integration* to calculate the velocity and displacement from the measured acceleration. The following subsections will explain how each of the g force (acceleration), RPM (velocity) and the stroke (displacement) are calculated. Please refer to Section 2.2 for the definition of each of these quantities.

2.5.1 G Force

The goal here is to calculate the average acceleration for each axis based on a number of sensor readings for that axis. The procedure for doing this is by:

1. Finding the minimum (Min) and maximum (Max) acceleration values for each set of readings.

2. Computing the average g for that axis, which is equal to $(Max + Min)/2$

The *Main g force* is computed as well, giving the machine's main acceleration based on the two axis determining its main motion. For example, if the main motion is determined by the X and Y axis, then the Main g Force is computed as follows:

$$Main\ g\ Force = max(\sqrt{x^2 + y^2} \forall x \in X, y \in Y) \tag{2.12}$$

2.5.2 RPM

The machine's RPM can be calculated from the FFT results. As described in section 2.4, performing an interpolation on the FFT is done to determine the fundamental frequency of the machine. The RPM is then calculated from the fundamental frequency f_0 as follows:

$$RPM(revolutions/minute) = f_0(revolutions/sec) \times 60(sec/minute) \tag{2.13}$$

2.5.3 Stroke

The stroke is the displacement of the machine. Since the machine can move in three axes, the stroke can be computed for each of these axis as follows:

$$(X, Y, Z)Stroke(mm) = \frac{(X, Y, Z)Average\ G\ Force \times 1000}{2 \times (RPM/60)^2} \tag{2.14}$$

Similarly, the machine's main stroke can be computed from the Main g Force (see section 2.5.1) as follows:

$$Main\ Stroke(mm) = \frac{Main\ G\ Force \times 1000}{2 \times (RPM/60)^2} \tag{2.15}$$

2.6 Frequency Analysis

Frequency analysis is done through two methods; FFT (Fast Fourier Transform) and FWT (Fast Wavelet Transform) which are both discussed in this section. Founded by the mathematician Joseph Fourier in the early 1800s, the Fourier Transform's idea is that a function (e.g a time domain based signal in the case of vibration analysis) can be constructed from a summation of sinusoidal functions with a continuous distribution of frequencies from zero to a suitable cutoff frequency. Based on that, transforming a signal into the frequency domain will identify all the components that contribute in creating that signal. This in turn helps identify sources of noise and/or frequencies that are caused by any damage in the vibrating object. In additon to the Fourier transform, there is a simple, but more restrictive version called the Fourier

series. The Fourier series method is mostly used for periodic repeating signals, and works by summing sinusiodal componvbents only at a discrete set of frequencies, which are integer multiples of a specified base frequency ($2\pi/t$, where t is the duration of the signal's period). The problem with the Fourier series in the case of machinery vibration, is that the frequencies are often not all integer multiples of a single frequency, this is why the Fourier transform proves to be more effective.

2.6.1 Fast Fourier Transform

The FFT algorithm was developed in the mid 1960s as an effective means for quickly mimicking the frequently changed radar signal spectrum of enemy ground-based antiaircraft missile targeting systems, so that multiple decoy signals could not be distinguished from authentic reflections [1]. Since then, the FFT algorithm has gained huge interest in the signal analysis field in general, and in vibration signal analysis in particular. The importance of the FFT algorithm lies in providing a way to transform time-varying signals from the time domain into the frequency domain on the spot. This provides a very useful method for analyzing the signal, since when plotted, the signal's frequency components are displayed. Prior to the development of the FFT algorithm, the primary method for displaying the vibrating signals was in the time domain using an oscilloscope. Other methods proved to be impractical especially since they required experience in tuning and adjusting the bandwidth filters in order to operate successfully.

Figure 2.9 depicts an example of an FFT spectrum of a rotating machine's vibration signal. The frequencies appear on the x axis, and their amplitudes on the y axis. It is worth noting that the plots have been scaled so that all the y values are ratios of the maximum y value.

In the example depicted by Figure 2.9, the frequency component with the highest amplitude in the X and Y directions is approximately $15Hz$. This normally would be the machine's running frequency. Other high frequency, low amplitude components appear as a sign of noise (particularly on the Z axis which is perpendicular to the machine's main movement). This noise can be caused by the machine itself (e.g a

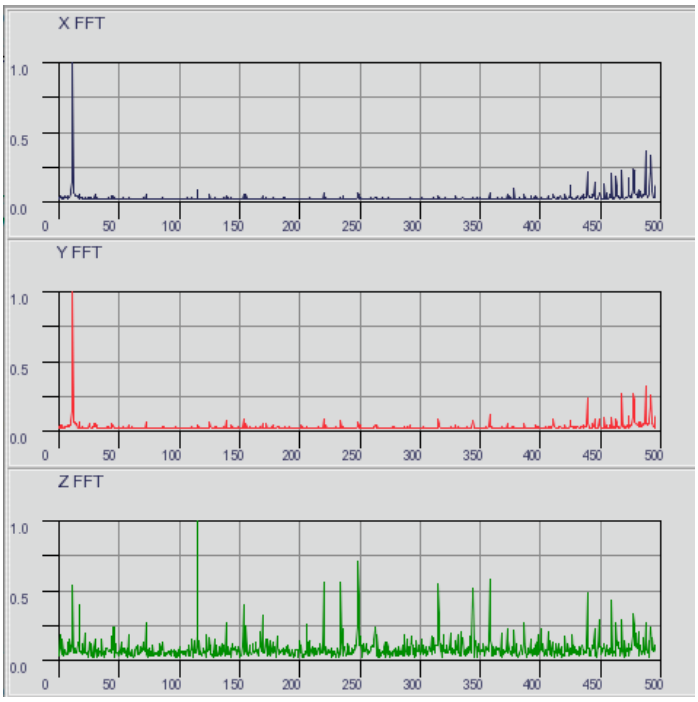


Figure 2.9: FFT plots for the X,Y and Z axes of a vibrating machine.

damaged component) or from the surrounding environment. This example demonstrates the power of the FFT algorithm in such an application in that it provides information that is very helpful in troubleshooting the rotating device.

Since its development, the FFT algorithm has introduced many new ideas in vibration analysis. One important aspect the FFT algorithm has introduced is the ability to diagnose vibration problems. Once FFT spectrums have been produced for various situations where a problem has occurred (e.g by a physical fault in the machine), future spectrums can be compared against them aiding in the diagnosis of the problem.

2.6.2 Fast Wavelet Transform

Wavelet Transforms are a powerful extension of the Fourier Transform [1]. Just like the FFT is a fast numerical computation of the Fourier Transforms, there exists a similar method for the Wavelet Transforms; the FWT (Fast Wavelet Trans-

form). The Wavelet Transform provides capabilites of isolating signal discontinuities with high resolution and detailed frequency analysis over long time windows yielding wavelets useful for *feature detection*, signal *noise-removal*, and *data compression*[1]. Currently, marketed machinery monitoring systems do not typically utilize *wavelet transforms*, but it is believed that the capabilities of next-generation machinery monitoring systems will be considerably advanced by their use, once wavelets and their advantages (mainly in vibration based trouble-shooting) are familiar to machinery vibration engineers[1]. Therefore, wavelet transforms are not a part of this thesis work, but are left as suggested future work.

2.7 Orbit Analysis

An *Orbit* is the two-dimensional depiction of the machine's movement. Orbits are in fact projectiosn of the *phase plots* on to the XY, XZ, and YZ planes. Hence, a machine with three degrees of freedom (in the X,Y, and Z directions) will produce three different orbits. Once the data is acquired, two types of analysis related to the orbits can be performed.

First, by plotting the data, *visual analysis* can be performed on the machine's movement. When using a filter (as described in Section 5 of this chapter), plotting the orbits of the filtered data as well as the orbits of the unfiltered data helps analyse the filter's performance, which in turn aids in adjusting the filter's parameters (e.g it's bandwidth and centre frequency) in order to reach a point where it is performing as required.

Figure 2.10 displays an orbit produced by the same rotating machine producing the FFTs in Figure 2.9. Part **a** displays the orbit based on unfiltered data, whereas part **b** displays the orbit based on the filtered data after passing through a 4th order Butterworth bandpass filter.

In addition to visually analysing the utilized filter's performance, orbit analysis can help in diagnosing mechanical problems in the machine. Figure 2.11 illustrates an example of a rotating machine, where a progressively worsening influence on the

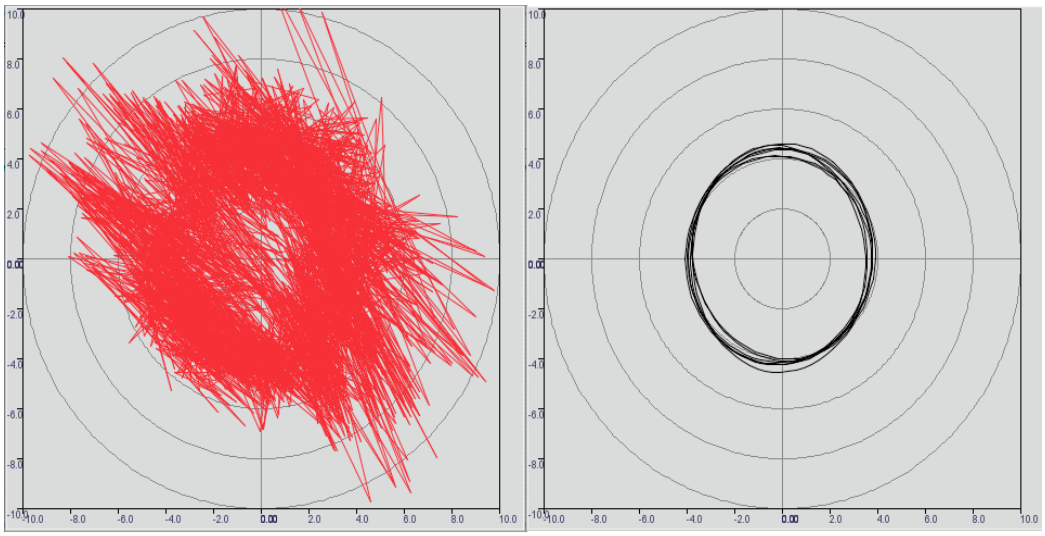


Figure 2.10: Filtered and unfiltered orbits of a rotating machine.

machine causes a progressively increasing static radial force on the rotor (and therefore on the bearings). Part **a** is the case where there is a normal radial load, the machine's main frequency is at N and its motion is synchronous and linear between the various parts of the machine. Part **b** is the same as part **a** but with a moderate increase in the radial load. Part **c** shows the case where there is a substantial radial load, the motion becomes nonlinear and a frequency component appears at $2N$. Finally, part **d** is where there is a very high radial load, the machine's motion is nonlinear, and there is a high frequency component at $2N$.

It is worth mentioning here that these orbits represent what are referred to as *Lissajous* plots. Lissajous plots are generated when graphing a system of parametric equations, which is exactly what is being done here. Each of the axis generates its own signal which is sinusoidal (please see section 2.8). Therefore, what is being plotted are two functions versus each-other, where their values are in the form:

$$x = Asin(at + \delta), y = Bsin(bt) \tag{2.16}$$

The appearance of the curves is highly senstive to the ratio a/b. When $a = b$, the resulting plot is an ellipse which is the normal case in vibrating machinery. As the

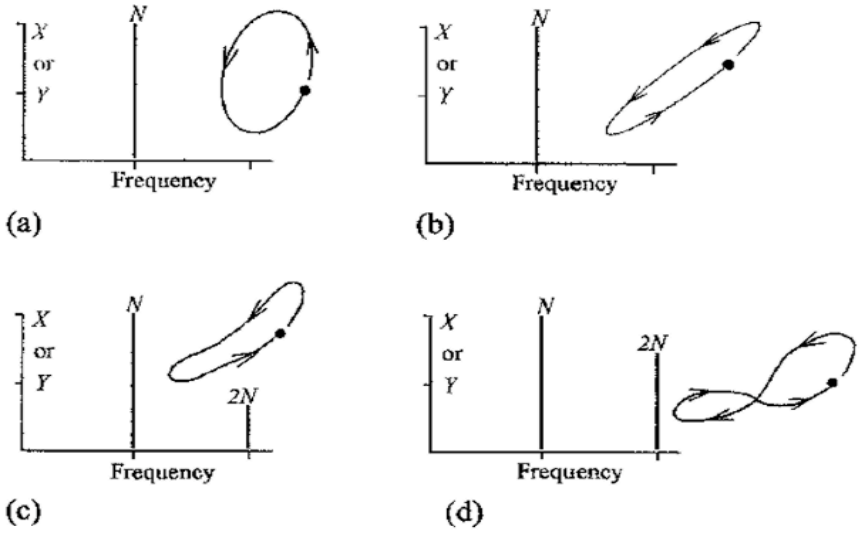


Figure 2.11: Filtered orbits and FFT for a rotating machine with increasing misalignments [1].

ratio increases, curves as in Figure 2.11 are generated, indicating mechanical faults in the machinery.

Second,*mathematical analysis* is performed by calculating the *phase angle* of the orbit (in rotating machines, the orbits are usually elliptical) as well as the *eccentricity* of the orbit. Sections 8 and 9 in this chapter explain more on how this is done.

2.8 Waveform Analysis

Another useful tool in vibration analysis, is plotting and visually analysing the waveforms. A *waveform* is the signal produced by plotting the data in a certain axis against time. In rotating machinery, the waveforms in general appear to be sinusoidal. Figure 2.12 depicts a waveform generated by a rotating machine.

It is clear from Figure 2.12 that the X and Y sinusoids are symmetric (like a cosine and a sine wave), in a sense that when plotting the Y values versus the X values (producing the orbits from the previous section) an elliptical plot is produced, describing the machine's main movement. Therefore, comparing the X and the Y

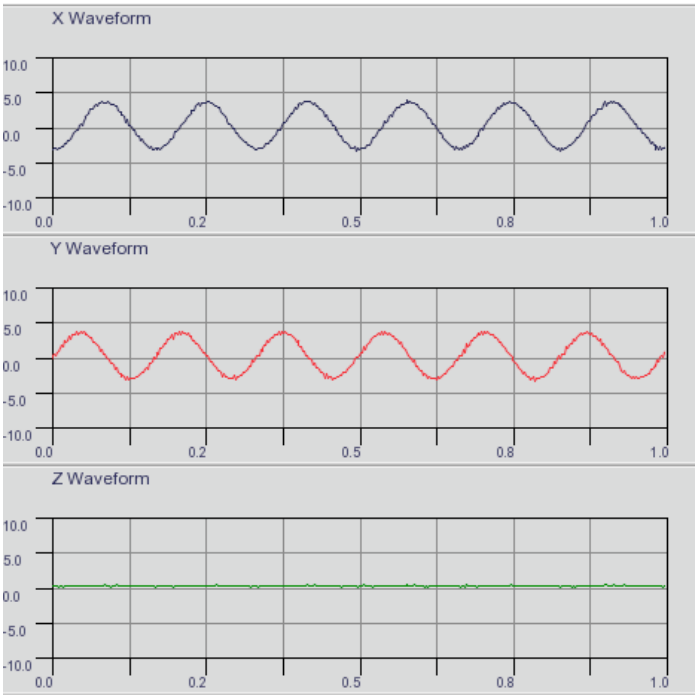


Figure 2.12: Waveforms generated by a vibrating machine in the X,Y and Z axes.

waveforms allows for a method of detecting problems in the machine.

Waveforms can help detect faults in the rotating machine. Although the waveforms will still appear to be sinusoidal even with noise, faults can be detected when seeing non-smooth peaks in the sinusoid. After discussing such situations with experienced engineers in the vibration analysis filed, it was determined that such irregularities appear usually as an “M” or a “W” shape at the peaks, indicating an additional frequency caused by noise. This noise will usually appear as a sign of a mechanical failure, such as a loose bolt. Figure 2.13, which is produced from the same rotating machine that the FFTs in Figure 2.9 and the orbits in Figure 2.10 are based on, depicts such a situation.

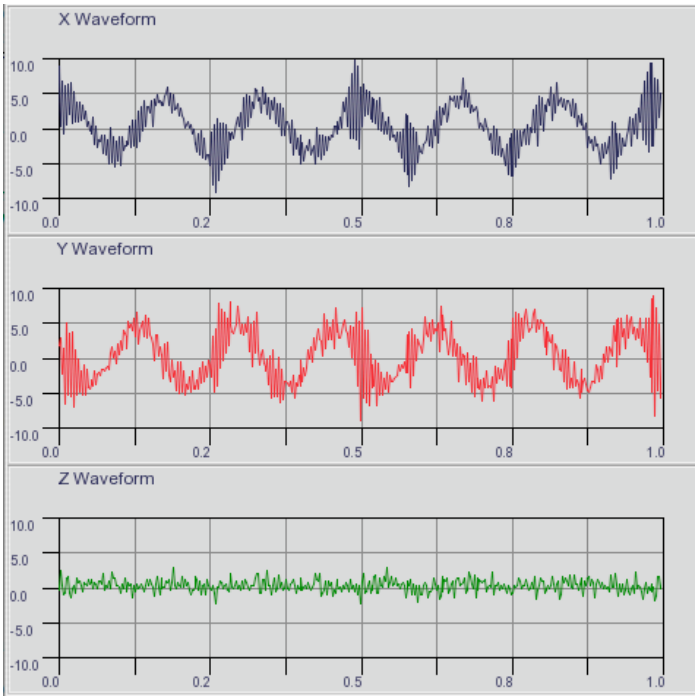


Figure 2.13: Waveforms generated by a vibrating machine with noise.

2.9 Ellipse Fitting

The motivation for performing ellipse fitting in our Vibration Analysis Tool is in providing a method to calculate the phase angles and the eccentricities of the orbits formed by the vibration of the rotating, especially since the orbits are expected to be elliptical. When more than one sensor is used to acquire data on different locations on the machine, it is very helpful to know the phase angles and the eccentricities of the ellipses created by each sensor's readings. It might be hard to distinguish visually between the various ellipses when plotting the orbits, so those two numerical values help in analysing the machine's vibration. This section will explain the method used to perform ellipse fitting, and how the phase angle and eccentricity quantities are determined.

2.9.1 Ellipse Fitting Method

Ellipse fitting on the acquired data is conducted by using the classical linear least squares method. An ellipse can be described by the conic equation:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{2.17}$$

Then, as explained in [5], the constants a, b, c, d, e and f are solved for using the least squares method on the given points. The minimum number of points required to fit an ellipse is five points [11], however, in order to get a more accurate result, more points are used in the Vibration Analysis Tool produced. Finally, the ellipse parameters (major and minor axes as well as distance between the ellipse's foci) are computed using the determined constants, and are thereafter used in computing the phase angle and the eccentricity.

2.9.2 Phase Angle

The ellipse's phase angle is the angle between the x-axis and the major axis of the ellipse. Therefore, An ellipse with its major axis lying on the x-axis has an angle equal to 0. Knowing the length of the semimajor and semiminor axes a,b respectively and the distance between the foci ($2c$) (please refer to Figure 2.14, the phase angle (θ), also referred to as the *angle of rotation* can be computed as follows:

$$\theta = \frac{1}{2} cot^{-1}(\frac{c-a}{2b}) \tag{2.18}$$

2.9.3 Eccentricity

The *eccentricity* (ε) of an ellipse is a measure of the shape of the ellipse. Mathematically, the eccentricity is the ratio of the distance between the centre of the ellipse and one of the foci (c), to the length of the ellipse's semimajor axis (a), or:

$$\varepsilon = c/a \tag{2.19}$$

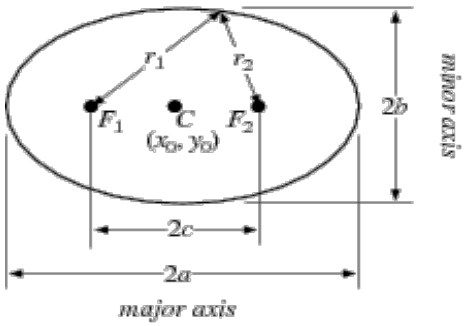


Figure 2.14: Ellipse example showing major and minor axes.

When $\varepsilon = 0$, the resulting shape is a circle (since the foci are at the centre, so $c = 0$), and when $\varepsilon = 1$ the shape becomes a parabola. Therefore, for an ellipse, the eccentricity value is expected to be $0 \leq \varepsilon < 1$. Please refer to Figure 2.15.

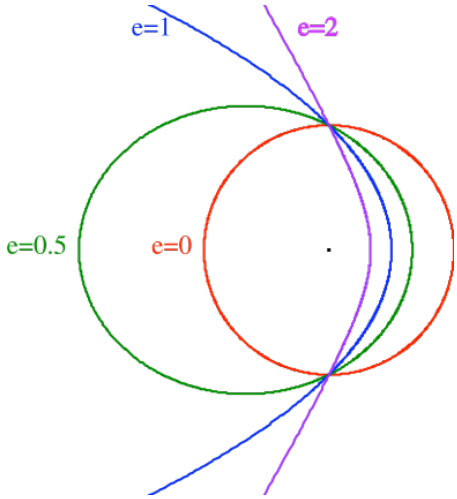


Figure 2.15: Examples of orbital trajectories with various eccentricities.

Chapter 3

Data Acquisition

The first step in vibration data analysis is acquiring the data. This chapter details the hardware used in performing this task. The software running on the chip was developed as part of a different thesis work and is therefore not discussed here.

3.1 Overview

There is a considerable variety of methods used to acquire and log vibrations. The methods and corresponding products available to accomplish data acquisition tasks constitute an ever changing field that parallels the rapid advancements in PCs and workstations [1]. This chapter provides a description of one such method used specifically for the vibration analysis tool developed as part of this thesis work.

The most common monitored vibration signals are *sound*, *displacement*, *velocity*, and *acceleration*. The vibration signals monitored as part of the developed Vibration Analysis Tool are accelerations in the X,Y and Z axes. The acquisition of these signals is done through a sensor called an *accelerometer*, and from its name, measures the change in acceleration. The signals are then passed through an Analog to Digital (A/D) converter located on a PIC micro-controller, and finally the digitized data is transferred through a Bluetooth device to the computer on which the Vibration Analysis Tool is running.

Figure 3.1 displays the sequence of electronics that the data travels through. It gets acquired by the sensors mounted on the vibrating machine, then transferred to the portable vibration analyser which in our case is a PDA, and finally downloaded to a PC where in depth analysis can be performed and data can be stored permanently. One major feature that the acquisition method described here adds is in the use of Bluetooth as the communication method rather than a wired serial connection. This increases the safety of the users by eliminating the wires attaching the analyser (which the user holds) to the vibrating machine.

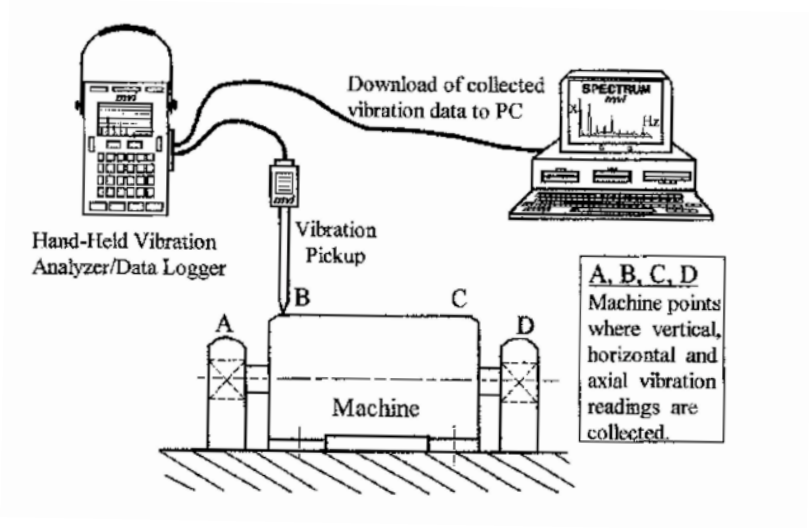


Figure 3.1: Portable machinery vibration analyser and data logger.

This chapter discusses the hardware used for the data acquisition, as well as describe the method used to calibrate the sensor readings.

3.2 Hardware Overview

A prototype of the hardware used as part of this thesis work is displayed in Figure 3.2. The three main parts of the hardware are: the PIC micro-controller shown in Part A, the Bluetooth device shown in Part B, and the accelerometer shown in Part C. The following three sections will detail each of these three units.

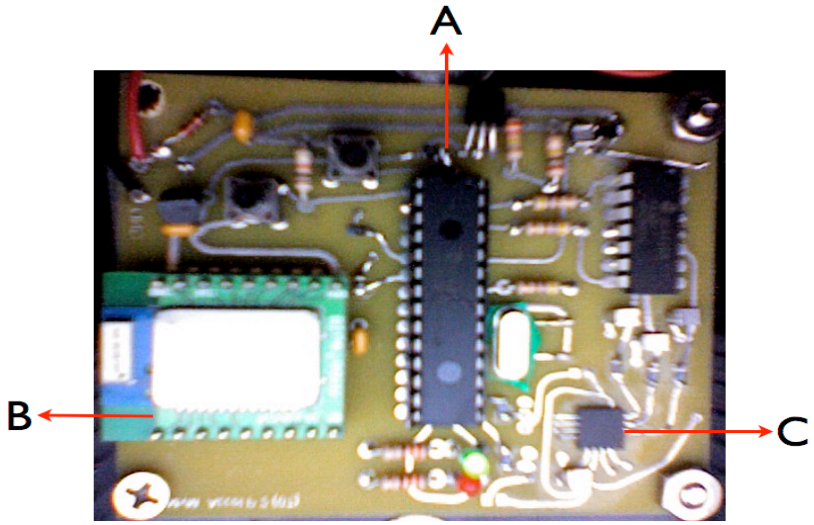


Figure 3.2: Data Acquisition Hardware Prototype.

3.3 PIC

The PIC micro-controller acts as a micro-computer on the acquisition device, receiving data from the sensor, passing it through an A/D convertor and finally sending it off to the Bluetooth device. Other vital operations occur on the PIC, but are not discussed as part of this thesis work. The main advantages of the PIC devices are in that they offer high computational performance at an economical price, as well as high-endurance. Additionally, for power sensitive applications, such as the data acquisition application, the PIC devices work best since they require low power to operate.

The exact PIC used as part of this thesis work is the PIC18F2423 which features a 12 bit Analog to Digital convertor, which allows for the coversion of data with low error (please see Chapter 8 for an in depth error analysis). Moreover, it features a serial communication capability which allows for using it with a Bluetooth device. Finally, it has a small size which helps in decreasing the total size of the acquiring device.

3.4 Accelerometer

An accelerometer (or an acceleration sensor), is a sensor used for measuring the acceleration of the object it is attached to. Accelerometers come in a variety of types, mostly differing by the number of axes they can obtain measurements in, and by the range of g values that they can pick up. An accelerometer is composed of an internal mass compressed in contact with a relatively stiff force-measuring load cell (this is usually a piezoelectric crystal) by a relatively soft preload spring. In response to dynamic loading, those piezoelectric load cells produce a self-generated electrical output at a very high impedance, which gets converted internally through electronics to a low impedance output suitable for data acquisition systems [1].

The particular accelerometer used as part of this thesis work is the MMA7261QT from Freescale Semiconductor. It is a low cost accelerometer that features signal conditioning, a 1-pole low pass filter, temperature compensation and g-Select which allows for the selection among four sensitivities (2.5g, 3.3g, 6.7g and 10g) [4]. Additionally, this sensor operates at a low power of 3.3V and can obtain measurements in three different axes.This sensor also features a Sleep Mode, making it ideal for handheld, battery operated devices like those used for the data acquisition. When the Sleep Mode is active, the device outputs are turned off, providing a significant reduction in the operating current. Finally, the sensor can operate under temperatures ranging between -40 and 105 degrees Celsius making it ideal for use in cold climates as well as hot ones.

3.5 Bluetooth Device

Bluetooth is a wireless technology that allows for the transferring of data between devices. Two main devices are involved: the Bluetooth device that sends the data, and the device that receives the data. Compared to other wireless communication technologies (such as Wi-Fi (IEEE 802.11)), the Bluetooth technology was found to work best for the data acquisition part of the Vibration Analysis Tool for many

reasons. First, it can operate over a distance of 10 meters or 100 meters depending on the class of Bluetooth devices used. Secondly, it is able to penetrate solid objects and it is omni-directional and does not require line-of-sight positioning of connected devices[2]. This is fairly important in such an application, since often times the vibrating machines are at locations that are hard to come close to, or are blocked by some objects depending on the site they are located at. Finally, Bluetooth technology is much cheaper compared to other wireless technologies, where the cost of Bluetooth chips is under 3 USD [2].

The Bluetooth device that was utilized as part of the hardware component in this thesis work is the *Bluetooth WRL-08461*. One feature of this device is that its Bluetooth stack is completely encapsulated. The end user just sees serial characters being transmitted back and forth. Pressing the 'A' character from a terminal program on the computer will cause an 'A' to be pushed out the TX pin of the Bluetooth module [13]. Also, this device operates on a low power or 3.3V, which is important especially since some of the machines will be monitored over a long period of time. Finally, these devices have been pre-programmed with a unique MAC address and unit name of 'SparkFun-BT' allowing for easy detection of the devices by the computer used for acquisition, as well as a UART interface of adjustable to a baud rate of 115,200 bps, meeting the data sampling requirement on the system.

3.6 Calibration

Data received from the sensor unit is scaled to the range of values available on the A/D convertor. The A/D convertor used has a 12 bit range, meaning values are in the range of $0 - 2^{12}$ or equivalently $0 - 4096$. However, data processed by the vibration analysis tool needs to be within the range of the selected g force.It is therefore important to calibrate each of the sensor units in order to obtain calibration parameters that are thereafter used to perform a scaling on the raw values so that they appear within the selected g-range. The following describes the method that the calibration is done through:

1. Three measurements at predefined g values first need to be taken for each of the three axes. The simplest measurements that can be acquired are at -1g, 0g, and 1g. To ensure the most accurate measurements possible, the sensor unit must be sitting on a surface completely level. It is important to note that by merely placing the device on the surface, one cannot guarantee that is it not experiencing a slight g force in the x and y directions. This is due to the way the device is packaged or any kinds of shifts from within its structure. One manual method to perform measurements that will return a reliable and accurate result is by rotating the device from +1g to -1g for each of the three axes [15]. The maximum value recorded during this rotation will correspond to a +1g force and the minimum value will correspond to a -1g force. Afterwards, assuming that the sensitivity is symmetric from zero to positive and from zero to negative, the sensitivity of the device can be calculated by dividing by 2. Now that the sensitivity is known, the 0g offset value can be calculated by adding the sensitivity to the minimum value or subtracting it from the maximum value. This value should be very close to the 0g offset value obtained by placing the device level. The following table shows the measurements as read from one of the sensor units for each of the available g ranges in the 3 different orientations.

Orientation	2.5 g	3.3 g	6.7 g	10.0 g
X (-1 g)	1503	1635	1835	1903
X (0 g)	2114	2090	2063	2055
X (1 g)	2696	2533	2284	2203
Y (-1 g)	1414	1569	1802	1877
Y (0 g)	2028	2033	2034	2033
Y (1 g)	2632	2483	2258	2183
Z (-1 g)	1290	1473	1757	1850
Z (0 g)	1970	1978	2008	2018
Z (1 g)	2665	2506	2272	2195

Table 3.1: Calibration measurements using gravity

2. After the calibration data has been acquired, the next step is to create the scaling functions. The goal here is to compute linear functions for each of the three axes measured for each one of the g-ranges. This is performed by applying a linear least squares fitting method on the three points measured above to obtain the best fit line's equation, which is then used as the scaling function. For instance, if this to be done for the x axis under 2.5 g mode, then given the three points $(1503, -1)$,$(2114, 0)$,$(2696, 1)$ from above, the goal is to obtain the best fit line represented by $f(x) = ax + b$. Performing a linear least square calculation leads to the best fit line represented by:
$f(x) = 0.001676x - 3.53$,
with a standard deviation of 0.0198, as displayed in Figure 3.3.

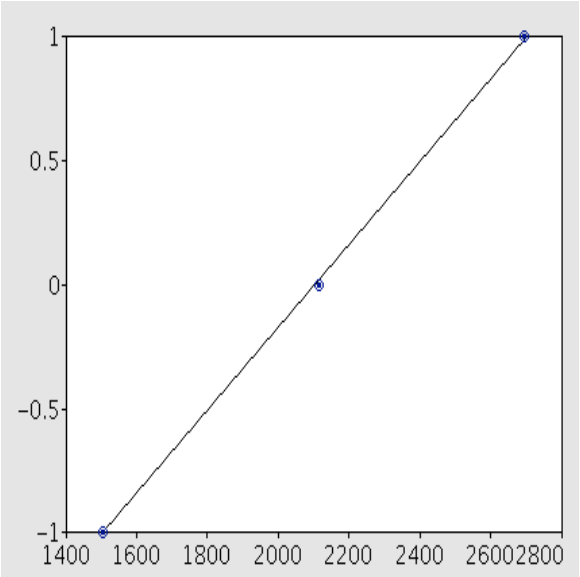


Figure 3.3: Best fit line for x axis measurements under 2.5 g mode.

Finally, it is important to note that the suggested method for calibration does contain errors that will be discussed in Chapter 8. A more precise measurement approach can be employed, such as building a testing rig that can operate on the 4 predefined g sensitivities. The sensors can then be calibrated on the testing rig in order to obtain the readings required for all three axes. One more point that should

be noted is the effect of the temperature on the readings. Chapter 8 will explain that more, but it is important that the sensors be calibrated at various temperature ranges. The temperature ranges can be defined by experiment, and the temperature sensor built into each unit can be used to obtain the surrounding temperature during the calibration.

Chapter 4

Requirements Specification

This chapter explains the requirements on the vibration analysis tool that is designed and implemented as part of this thesis work. The requirements represent a contract between the developer and the client, and are used throughout the software life cycle.

4.1 Overview

The vibration analysis tool that is designed and implemented as part of this thesis work has many functions. Its main functions are in connecting with the sensors, reading the acquired values, performing signal conditioning on the data, plotting the required graphs, logging the data and generating reports. In addition to reading from the sensors, it allows for the playback of the data that has been perviously logged into files. The tool is to have two different interfaces, with the same functionality, to run on two devices with different display areas. The first is a PDA which has a small display area, and the other is a PC with a larger display area. Therefore, in terms of the interface, two designs have to be made.

The requirements document is an essential document acting as a contract between the developer and the client. Figure 4.1 is an illustration of the *Waterfall Model* used frequently in software engineering. The design and implementation of the Vibration Analysis Tool produced as part of this thesis work is based on this model. As can be

seen, the requirements stage is a critical stage in the life cycle of the software, and is referred to throughout all its successive stages in order to verify the satisfaction of the requirements. This chapter discusses the application requirements.

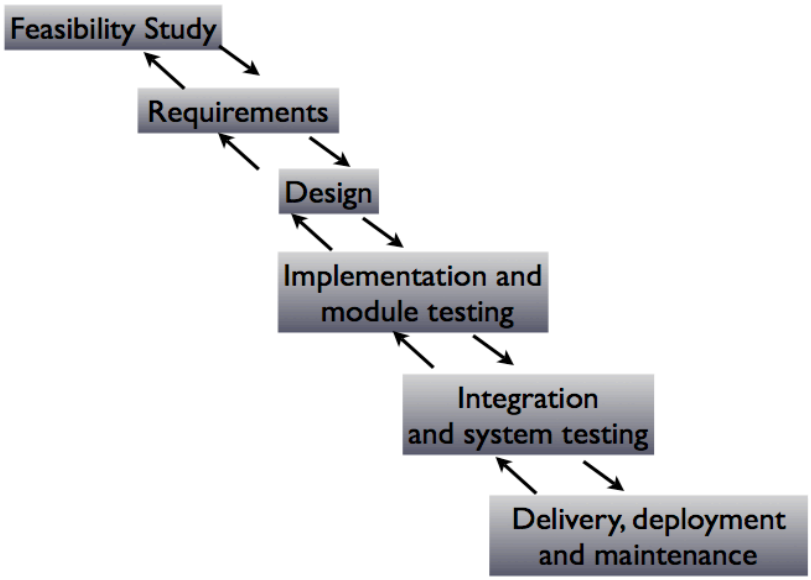


Figure 4.1: A waterfall process with feedback.

4.1.1 User Classes and Characteristics

Various users will have access to the developed Vibration Analysis Tool. These users differ based on their frequency of use of the tool, the subset of functions they access, and their technical expertise. Mainly, the users will either be technicians acquiring data on site, or expert engineers who would like to look at the acquired data more in depth and produce the reports containing recommendations to the client. Based on that, the tool will have to be designed in such a way that allows the technicians to acquire data easily, at the same time allowing expert engineers to access more detailed functions. Such detailed functions include adjusting the filter parameters, changing the predefined acceptance levels used in the reports, adding additional bearings with new diameters and producing the final reports after making certain expert decisions.

4.1.2 User Documentation

In addition to the Vibration Analysis Tool software, user documentation is provided. This documentation is in the form of a user manual detailing the various functions in the tool and how they are used. In addition, the user manual explains how the various plots are to be read, and how the reports are to be generated and interpreted.

4.1.3 Assumptions

The following is a list of assumptions that the design of the Vibration Analysis Tool is based on:

1. The tool shall only connect to at most eight sensor units at a time.
2. The sensor units must operate normally at any temperature between -40 and 50 degrees Celsius.
3. The sensor units must operate normally under the circumstances expected at sites where the analysis is performed.
4. The sensor units must detect g forces anywhere between 0 g and 10 g with a resolution of 0.01 g.
5. The sensor units must have the capability to read the battery level on the unit and send it to the computer with the vibration analysis tool.
6. The sensor units must have the capability to read the temperature level surrounding the unit and send it to the computer with the vibration analysis tool.
7. The sensor units must allow for the changing of the g select upon receiving a command from the computer with the vibration analysis tool.
8. Data on the sensor unit is to be sampled at a rate of 1000Hz.
9. The PDA interface is designed for a specific PDA.

4.2 Functional Requirements

An application's functional requirements are those describing its behaviour and are most likely implemented as functions in the implementation stage. The following are the functional requirements related to the Vibration Analysis Tool. The requirements and have been separated into sections based on the feature they deal with.

4.2.1 Communication Requirements

The following are requirements pertaining to the communication with the sensor units.

1. The application shall discover all available sensor units within the range of the bluetooth device.

2. The application shall connect to the selected devices.

3. The application shall read each device's ID.

4. The application shall read data coming from each of the sensor units.

5. The application shall send the selected g value back to the sensor unit.

6. The application shall read the battery level coming from the sensor unit.

7. The application shall read the temperature level from coming from the sensor unit.

4.2.2 Mode Requirements

The following are requirements related to the two modes that the application can be in.

1. The application shall allow the user to choose between two different modes: *File Simulation* and *Data Acquisition*.

2. In the *File Simulation* mode, the application shall allow the user to a list of files to play back data from.

3. In the *Data Acquisition* mode, the application shall allow the user to choose which devices to connect to from the available bluetooth devices.

4.2.3 Signal Processing Requirements

The following are requirements that deal with the signal processing part of the application. They describe the type of calculations that need to be performed on the acquired data before it is displayed.

1. The application shall perform use the sensors' calibration information to calibrate the data so that it appears within the selected g range.

2. The application shall pass the raw data through a DC filter prior to plotting it in order to eliminate the effect of gravity on the sensor unit.

3. The application shall apply a filter on the raw data in order to eliminate noise.

4. The application shall perform an FFT on the data in order to determine the fundamental frequency.

5. The application shall perform an ellipse fitting on the data in order to determine the phase and eccentricity of the machine's elliptical motion.

6. The application shall perform a calculation to determine the machine's RPM.

7. The application shall perform a calculation to determine the strokes in all three axes as well as the main stroke.

8. The application shall perform a calculation to determine the g force in all three axes as well as the main g force.

4.2.4 Logging Requirements

The following are requirements describing how the data is logged on the machine, and how it is thereafter read. The logging involves storing the raw data, as well as logging the acquisition information, such as the customer, the machine number, etc.

1. The application shall log the calibrated unfiltered data into a file.

2. The application shall allow the user to determine when the logging should start and when it should end.

3. The application shall create a separate file each of the used sensor locations for logging its data.

4. The application shall log information related to the acquisition into a separate file. This information is to include at least the following: Customer, machine model, date and number of bearings on the machine.

5. The application shall log information related to the sensor units used. This information should include at least the following for each used sensor: Sensor ID, location on machine, orientation (on the top or on the side of the machine), battery level and temperature level.

6. The application shall log the start and end times of the acquisition.

7. The application shall log the g force level used throughout the acquisition.

4.2.5 Display Requirements

The following requirements apply to how the data is displayed on the screen. The display of data can be in two forms: graphical and numerical.

1. The application shall display plots of the machine's motion in the XY, XZ, and YZ planes.

2. The application shall allow for displaying the orbits based on both the filtered and unfiltered data.

3. The application shall display small views of the orbits of all the locations used to perform the acquisition at the same time.

4. The application shall display values related to all the locations used to perform the acquisition at the same time.

5. The application shall display plots representing the FFTs in all three axes for each of the sensors. The frequencies displayed shall be of a minimum of 10Hz and a maximum of the Nyquist frequency given by half the sampling rate.

6. The application shall display plots of the signals (waveforms) in all three axes for each of the sensors.

7. The application shall display the machine's RPM computed from each of the sensors.

8. The application shall display the stroke in all three axes for each of the sensors, as well as the average stroke.

9. The application shall display the g force in all three axes for each of the sensors, as well as the avergae g force.

10. The application shall display the phase and eccentricity after performing the ellipse fitting on each of the sensor locations.

11. The application shall display the information related to the acquisition. This includes the customer, the machine model, the date, the start and end times, and the number of bearings.

4.2.6 Graphical User Interface Requirements

The following are requirements pertaining to the Graphical User Interface, and describe the types of menus, controls and messages that are to be used.

1. The application shall allow the user to choose the g force the machine is operating on.

2. The application shall allow the user to pick a sensor to display its enlarged orbit plot, FFT plots and waveform plots.

3. The application shall allow for the pausing and resuming of the data display.

4. The application shall allow for initiating and stopping the recording of the data.

5. The application shall allow for the adjusting of the filter bandwidth.

6. The application shall allow for the adjusting of the FFT frequency range.

7. The application shall allow the user to change the orientation of the orbit plots between the XY, XZ and YZ planes.

8. The application shall allow the user to choose whether the enlarged orbit shall display filtered or unfiltered data.

9. The application shall allow for adjusting the centre frequency used for the filter.

10. The application shall allow the user to choose the unit used to display the stroke (mm or inches).

11. The application shall allow the user to view generated reports directly from within the application.

4.2.7 Reporting Requirements

The application must allow for the generation of three types of reports as required by the client. The following is a description of these reports:

Single Point Report

Such a report is generated for each of the mounted sensors, providing in depth analysis on each of the individual measuring points. An FFT plot, a waveform plot and an orbit plot (with both filtered and unfiltered data) is generated along with a summary of the numerical values for that specific location.

Orbit Summary Report

This report must contain orbit plots of all the mounted sensors, as well as their numerical values. The orbits shall appear in their filtered state, allowing the user to determine whether or not the machine is performing normally. The machine's behaviour is deemed *normal* if the orbits have the same size, the left side of the machine mirrors the right side, and the overall acceleration levels are within their expected range.

Tuning Report

This report summarizes the vertical and the horizontal strokes and g forces for the entire set of sensors mounted to the machine. It is mainly used to determine any deviations between the measured data and the expected machine behaviour. The results of the deviations are used to provide tuning recommendations to the client based on physical principles that apply to any vibrating machine.

The following are requirements that relate to the reporting part of the application.

1. The application shall allow for the creation and storage of a single point report for each sensor location.

2. The application shall allow for the creation and storage of an orbit summary report summarizing the behaviour that all of the mounted sensor units acquired.

3. The application shall allow for the creation and storage of a tuning report detailing the machine's behaviour and providing recommendations for adjusting

the machine.

4. The application shall allow for the adjustment of the values used to generate the turning report before the report is actually generated. Such values include the bearing diameter, the friction compensation, the inclination angle. and other important parameters.

5. In the tuning report, the application shall allow the user to manually enter values for missing sensors based on their location and the readings from the other sensors.

6. The application shall allow for the viewing of the reports if they have already been created, and the creation of reports otherwise.

7. The application shall generate reports in the formats predetermined by the client.

8. The application shall include a header section on each of the reports with the customer, machine, date and time information as well as the company logo.

4.3 Nonfunctional Requirements

An application's non-functional requirements are those not related directly to its behaviour, but are more concerned with its properties or characteristics. The following are the non-functional requirements that have been found to be related to the Vibration Analysis Tool.

4.3.1 Performance Requirements

Performance requirements are most important in hard real-time systems, where failing to meet a time requirement causes the entire system to fail. This is not exactly the case in vibration analysis, since not meeting a deadline will not cause a system failure,

however, it may cause unreliable data. For this reason, the following requirements are necessary:

1. The application shall generate plots and display data read from the sensor at a frequency defined by the user and not exceeding 1000Hz.

2. The application shall log **all** data read from **all** sensors.

4.3.2 Platform Requirements

The following are requirements related to the operating system and devices that the application will run on.

1. The application shall operate on Linux (for the PDA application) and Windows (for the PC application).

2. The application shall run on both a PDA device selected by the client, as well as a PC (laptop or desktop) with no less than a 12" display.

4.3.3 Safety Requirements

Since the vibration analysis tool is not a safety critical application, not many requirements exist in terms of safety. The following is a requirement on the connection with the sensor units.

1. The application shall connect to the sensor units via bluetooth. This will allow for a wireless connection that increases the safety of the person acquiring the data, since they will no longer be holding a device that is connected by a wire to the vibrating machine.

4.3.4 Maintainability Requirements

The maintainability requirements focus on both the maintenance of the application as well as the ability of adding features later on.

1. The application shall be designed in a manner allowing for easy maintenance, as well as in an easy way to add features in the future.

4.3.5 Reliability Requirements

Informally, a software is reliable if the user can depend on it [7]. The reliability in the vibration analysis tool comes from the reliability of the data being read. The following are requirements concerning the reliability of the application.

1. The application shall read the data coming from the sensor in a reliable manner, meaning that data shall not be dropped or misread.

2. The application shall produce reliable plots as determined by the engineers at the company.

3. The application's signal processing calculations shall deliver reliable results as determined by the engineers at the company.

4. The reports produced by the application shall contain reliable results as determined by the engineers at the company.

4.3.6 Usability Requirements

A software system is usuable if its human users find it easy to use [7]. As discussed earlier, more than one type of users may be using the application, and therefore, the application should be usable for every type. The following are requirements on the usability of the application.

1. The application shall have a Graphical User Interface (GUI) and shall use controls that allow the user to access its various functions.

2. The application's interface shall be easy to use as determined by its various types of users.

3. The application shall allow for the display of text (on labels, button, etc.) in any of the predefined languages.

4. The application shall use fonts that are legible to an average user.

5. The application shall use colours that convey meanings to the user while maintaining the look of the application.

6. The application shall use consistent colours, fonts and designs.

Chapter 5

Software Design

5.1 Overview

The step after the requirements specification of a system, is its design. Software design is a fundamental step in the software development process as it acts as a bridge between the requirements and the implementation of the system. The main goal of software design is to decompose the system into parts, assign responsibilities to each part, and ensure that the parts fit together to achieve the global goals of the system [7]. This chapter discusses the software design of the vibration analysis tool. It starts by explaining the method used for structuring the design, and thereafter discusses the main system modules and how they interact.

5.2 Design Method

The first step that was taken in order to design the system was to determine the system tasks. A flowchart was used in order to illustrate the flow of tasks in the system and is depicted in Figure 5.1.

The next step was to divide the system into modules (or components) keeping in mind the software design principle of *information hiding*, where each module hides internally information that can be kept secret from all the other modules. This

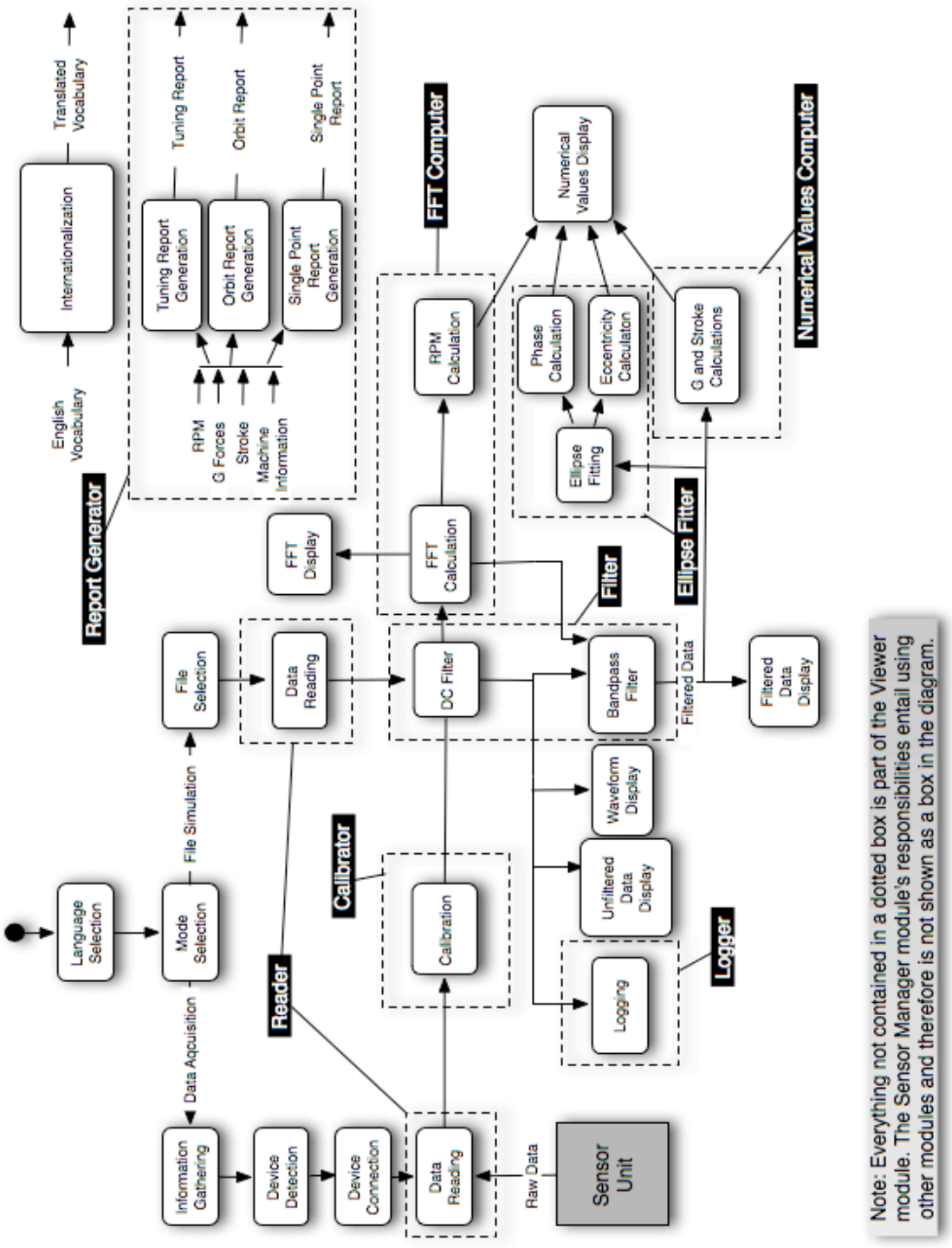


Figure 5.1: Data flow in the system.

technique ensures that the system is structured in such a way that allows for future changes and therefore makes it more maintainable and easy to extend. If a change is to occur in the method that data is logged for example, only the *logger* module (refer to Section 5.3.4) would have to change its *log* function.

Additionally, since the software to be developed has a significant amount of user interaction, one well-known software architecture to be used in this case is the *model-view-controller* (MVC) architecture. MVC is composed of three components: the *mode*l, which models an object in the "real world", the *view*, which displays the model to the user, and the *controller*, which communicates with the user and controls the other two components [7]. This type of architecture is beneficial in that it makes multiple views possible for the same model, and changes can be made to the view without affecting the model, and vice versa. Figure 5.2 depicts the MVC architecture.

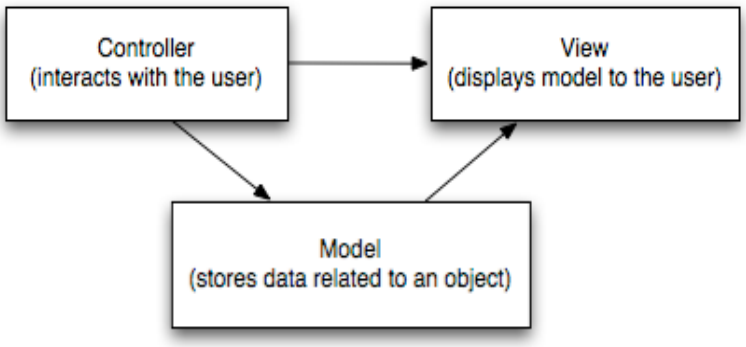


Figure 5.2: The model-view-controller architecture.

In the vibration analysis tool designed here, the model represents a location being monitored on the vibrating machine. This model stores any information relevant to that location such as its g forces in all three axes, the strokes calculated from the g forces, the resulting ellipse phase, its filtered and unfiltered data and so on. The view is basically the main frame of the application which acts as a listener for the model and updates its components (the waveform plots, the orbit plots, the FFT plots and the numerical values display) based on the model. As for the controller, often times in such user interactive applications, it ends up being the same component as the view

(Please refer to Figure 5.3). This is the case in the vibration analysis tool developed, where the main frame of the application contains an options component which allows the user to adjust specific settings such as those related to the filter (its bandwidth and centre frequency) or select whether filtered or unfiltered data is to be displayed. Other options include the selection of the g range, the orientation (Y vs. X, Y vs. Z, X vs. Z) to be displayed on the orbit plots, the maximum frequency to be displayed on the FFT plots, and the unit used to display the stroke values in.

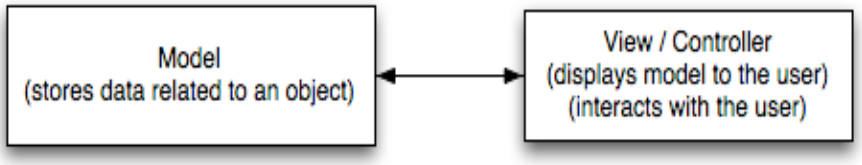


Figure 5.3: Alternate model-view-controller architecture.

The following section lists the main system modules and their responsibilities.

5.3 Main Modules

The following are the main system modules. Each of these modules encapsulates some of the tasks shown in Figure 5.1. In the implementation phase, each of these modules contains sub-modules (represented as classes).

5.3.1 Reader

The *Reader* module is responsible for reading data from the handlers. A handler can be either a file (in the File Simulation mode) or a serial connection (in the Data Acquisition mode). The left part of Figure 5.4 illustrates the main loop of the reader module. As soon as the acquisition or simulation is started (on the event *run*), the reader module starts reading data from the handlers and putting it in queues (one for each handler) for the sensor managers (see Section 5.3.2) to pull data from later. The *Reader* module stops reading on the *stop* event.

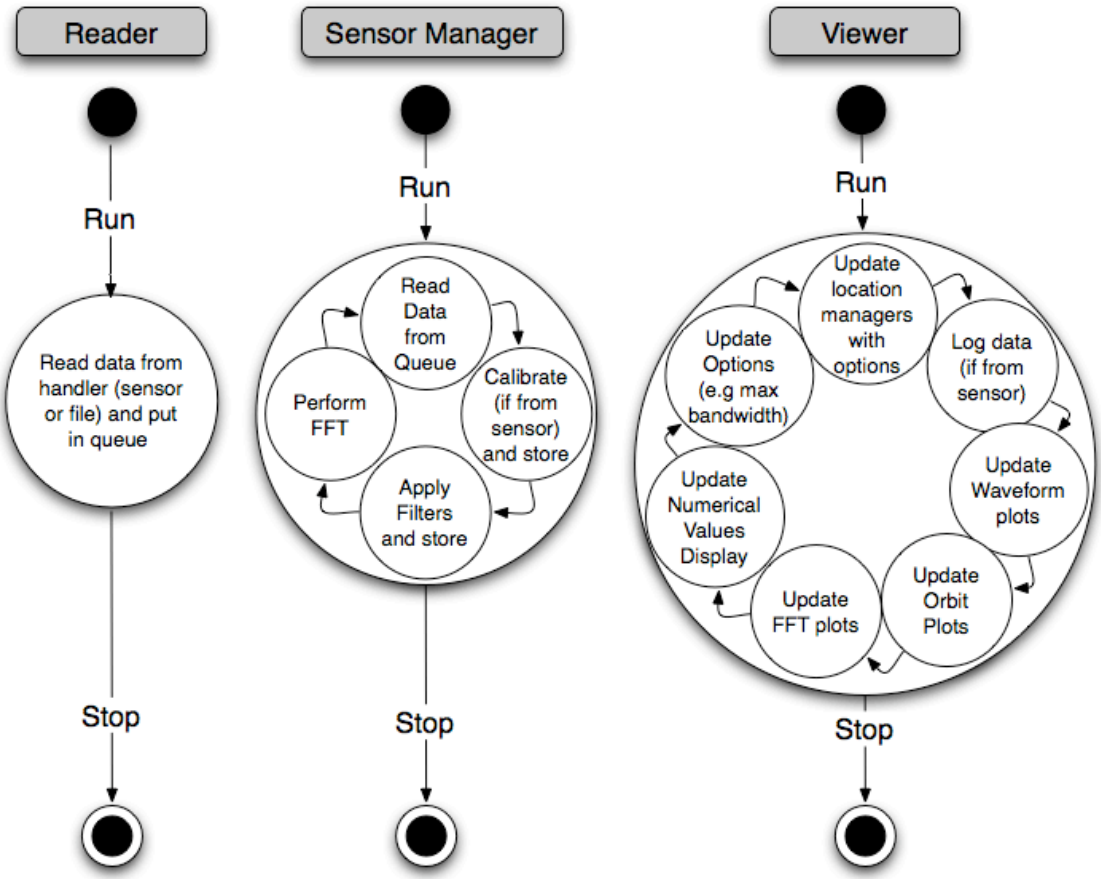


Figure 5.4: Depiction of the main system loops.

5.3.2 Sensor Manager

For each handler object created, a *Sensor Manager* module exists. The *Sensor Manager* represents the *view* in the MVC architecture described in Section 5.1. Its main responsibilities are in reading data from the queues, calibrating the data if being read directly from the sensor, communicating with the *Filter* module in order to filter the raw data, communicating with the *FFT* module in order to perform an FFT computation on the raw data, and storing both filtered and unfiltered data for the *View* module to access. Figure 5.4 displays in the middle the main loop of the *Sensor Manager* module. Its *run* and *stop* events are synchronized with those of the *Reader* and *Viewer* modules.

5.3.3 Viewer

The *Viewer* module acts as both the *view* and the *controller* in the MVC architecture. As the *view*, and based on a timer that issues events, it updates the waveform, orbit and FFT plots as well as the numerical values display and the options panel (e.g the maximum value allowed for the bandwidth setting) with the most recent data stored in the *Sensor Manager* modules. As the *controller*, it receives information from the user when changing settings in the options panel. and updates the *Sensor Manager* modules accordingly. Such information includes the bandwidth and the centre frequency used for the filter. Figure 5.4 illustrates on the right the main loop of the *Viewer* module which is synchronized with the *Reader* and *Sensor Manager* modules on the *run* and *stop* events. It is also worth noting that this is the only module that contains GUI related functionality which is abstracted away from all other modules.

5.3.4 Logger

As its name implies, the *Logger* module's responsibility is in logging the data. This module has been kept separate in order to facilitate for changes in the future which might include changes in the method data is logged. The *Logger* module exists only

for data that is being read directly from the sensors. One *Logger* module exists for each sensor handler object.

5.3.5 Calibrator

The *Calibrator* module exists only in the data acquisition mode where data is read directly from the sensor. This module has been kept separate in order to allow for easy change in the calibration method without affecting other modules. One such module exists for each sensor handler based on that sensor's calibration parameters and the g range being used for aquisition. Each *Calibration* module interfaces with the *Sensor Manager* module by receiving non-calibrated data from from it and sending back calibrated data.

5.3.6 Filter

The *Filter* module contains sub-modules each representing a different filter type. It interfaces with the *Sensor Manager* modules by receiving unfiltered data and sending back filtered data. This separation of concerns between the modules allows for changes in the way the filters are implemented without affecting the *Sensor Manager* modules.

5.3.7 FFT Computer

The *FFT Computer* module is responsible for applying an FFT computation on the unfiltered data that is received from the *Sensor Manager* modules. It is also responsible for computing the fundamental frequency based on the data set it receives, as well as the RPM (which is based on the centre frequency), and sending back these two values along with a list of frequencies and their absolute values for the *Viewer* module to access later for generating the FFT plots.

5.3.8 Numerical Values Computer

This module is responsible for computing the g forces and the strokes based on the filtered data it receives from the *Viewer* module. The computed values are then sent back to the *Viewer* module so that they can be stored in the *Sensor Manager* modules and displayed on the numerical values display.

5.3.9 Ellipse Fitter

The *Ellipse Fitter* module is responsible for performing an ellipse fitting on the filtered data it receives from the *Viewer* module. Additionally, it is responsible for computing the phase and eccentricity of the ellipse and sending these two parameters back to the *Viewer* in order to display them on the numerical values display and store the phase parameter in the *Sensor Manager* modules for the report generation.

5.3.10 Report Generator

The *Report Generator* module contains three sub-modules, each responsible for generating a different type of report. The Single Point Report module which generates a report for each monitored location on the machine, the Orbit Summary Report which generates a summary report for all the monitored locations, and the Tuning Report which based on certain parameters entered by the user, generates a report containing recommendations for tuning the machine in order to achieve better performance.

5.4 Main Module Interaction

Figure 5.5 illustrates how the main modules discussed above interact. It is worth mentioning here that the *Viewer* module, which is basically the user interface, also receives events from the user such as initiating (or stopping) the acquisition or simulation which correspondingly initiates (or stops) the *Reader* and *Sensor Manager* modules' main loops. Such events have not been included as part of the diagram and only the data transferred between the modules is shown.

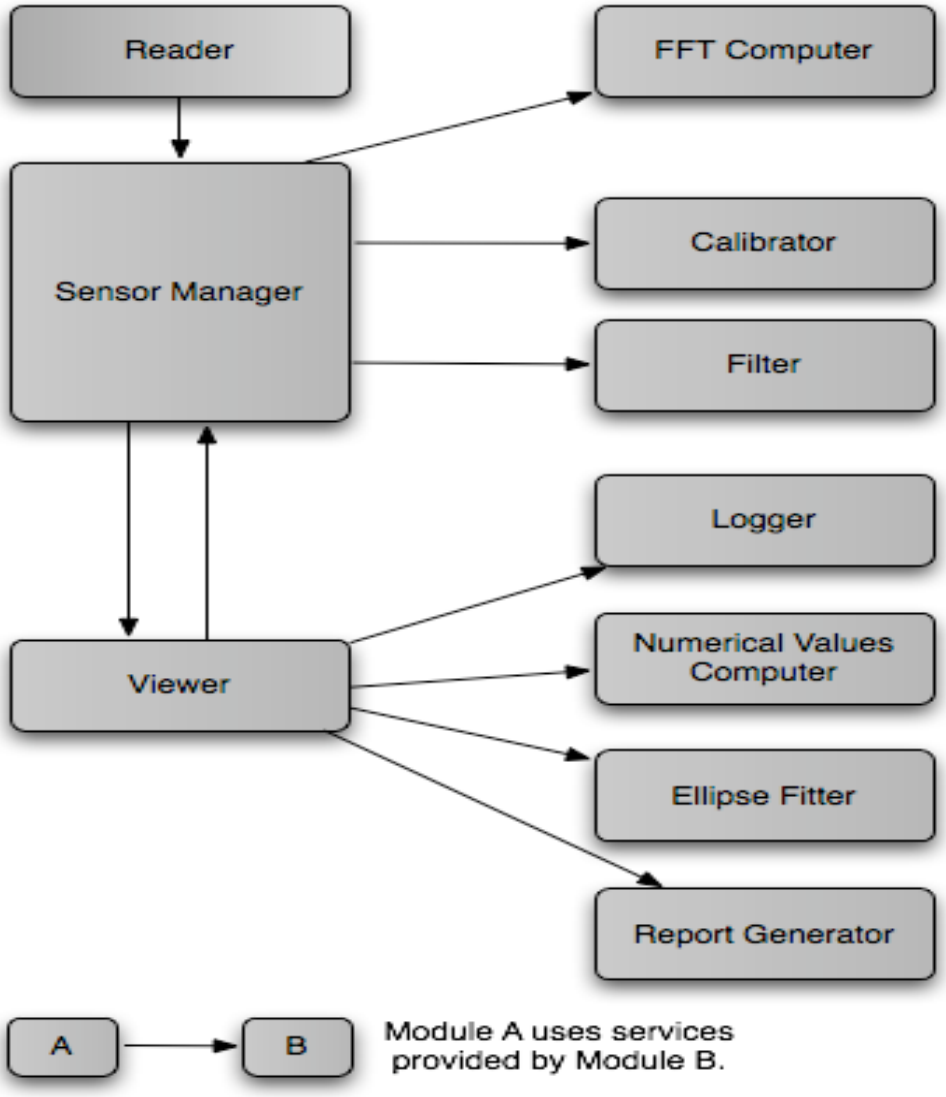


Figure 5.5: Module interaction diagram.

Chapter 6

Software Implementation

6.1 Overview

This chapter discusses the implementation phase of the vibration analysis tool's llfe-cycle. It starts by a section discussing the considerations taken throughout the implementation phase to ensure the development of a high quality software. Then, it lists the software libraries that were used as part of the implementation. Afterwards, the implementation of the main software modules discussed in Chapter 5 along with some helper modules is discussed.

6.2 Implementation Considerations

The following software qualities were considered throughout the implementation phase:

- **Portability**: A software is considered portable if it can run on different environments. The software was developed using the *Python* programming language along with some libraries explained in Section 6.3, which are all guaranteed to run on both Windows and Linux platforms as requested by the company. Moreover, the software is portable in that it can run on different hardware as well; on the PDA and on a PC.

- **Performance**: A software's performance is linked to its usability and reliability. The better the software performs, the more usable and reliable it is. In terms of vibration analysis, performance is a major software quality that needs to be considered. The values are acquired at a rate of 1000 Hz and are to be processed by the vibration analysis tool at a rate that will guarantee all the data being processed. Therefore, the software was implemented in such a way that will allow for that. Most importantly was the use of *pyrex* in order to convert some parts of the program into a C implementation. The parts that are converted are those that are called most often and involve the data processing (unpacking the data coming from the sensors, storing the data in a circular queue, applying filters on the data, etc).

- **Usability**: A software is considered usable if its users find it easy to use. After discussing user interface aspects with the personnel who would be using the tool most often at the company, an interface for the software was designed. The interface was implemented in such a way that allows for easy access to various commands. Moreover, it provides a clear view of the plots that display different forms of the acquired data. The reports are also generated in a format approved by the company, and in the case of the *Tuning Report* the user is allowed to enter all the parameters required and preview the results before actually generating the report. Finally, the addition of internationalization allows for the selection among various languages as requested by the company in order to make the software usable in different countries.

- **Maintainability**: Studies have shown that the majority of time spent on maintenance is spent on enhancing the product with features that were not in the original specifications. Therefore, the software was developed in order to allow for easy maintenance. This was done primarily in the use of modularization in the design phase, splitting the software responsibilities among various modules each with its own responsibility. Moreover, the use of the information hiding principle allowed for changes to be done on modules without affecting the rest.

- **Correctness, Reliability and Robustness**: These qualities are linked together and are very important in guaranteeing accurate results. Joined together they characterize a software quality that implies that the application performs its functions as expected. The software developed is correct in that adheres to its specifications, implementing all its functions as specified in the requirements phase. It is reliable in that its results are correct and that it behaves as expected over a long interval of time. Finally, the software is robust in that it behaves reasonably even in circumstances that were not anticipated in the requirements specification. For example, in the case that the connection is lost with one of the sensors, the software will notify the user and normally keep on running with the other sensors without causing a crash to occur. In general, the implementation was conducted in such a way that guarantees robustness. Try-Except statements were used only when needed, and when used exceptions were explicitly defined. Moreover, in the model-view-controller architecture used, when the model (the Sensor Manager module) is changing parts of the view (the Viewer module), events are used in order to guarantee that they occur correctly. Finally, when using threads, it was kept in mind not to have any shared state between the threads, and that data structures used with threads are accessed in a safe manner. This is the case of the Circular Queue module which will be explained in the following section.

- **Reusability**: One of the main goals of object oriented programming is to achieve reusability. Since the developed software was implemented in *python*, object oriented programming was used and reusability was ensured through the creation of classes that can be used in both the PDA and the PC applications. A great amount of code was reused between the two applications allowing for easy maintenance without having to change the code twice. The only module that is not completely common between the two applications is the *Viewer* module, and even here *inheritance* was used in order to keep as many things as possible common and therefore reusable.

- **Understandability**: The developed software is easy to understand primarily due to its modularity. In addition, the code is well documented and commented to allow for developers to continue working on it in the future.

6.3 Software Libraries Used

The following is a list of the main software libraries used as part of the developed application. A brief description accompanies each library.

- Threading: This module provides a high level threading interface providing accessibility to threads and their methods. It is used in both the *Reader* module and the *Device Discovery* helper module.

- PyBluez: This module creates python wrappers around system Bluetooth resources to allow for easy and quick creation of Bluetooth applications. It works on GNU/Linux and Windows XP (Microsoft and Widcomm Bluetooth stacks). It is used in both the *Reader* module and the *Device Discovery* helper module.

- PySerial: This module encapsulates the access for the serial port. It is used by the *Reader* module in the case of the software being run on Mac OS, since PyBluez is not available there.

- NumPy: This is the fundamental package needed for scientific computing with Python. It contains a powerful N-dimensional array object, basic linear algebra functions, and basic Fourier Transforms. It is used in the *Sensor Manager* module, the *Viewer* module, the *Filter* module, the *FFT* module, the *Ellipse Fitter* module, and the *Report Generator* module.

- gettext: This module provides internationalization (I18N) and localization (L10N) services for the python application. It is used in the *Viewer* module only for translation purposes of the labels and other controls with text.

- wxPython: This module is a blending of the wxWidgets C++ class library with the Python programming language. It provides GUI abilities and is therefore used in the *Viewer* module only.

- The Python Imaging Library (PIL): This library adds image processing capabilities to the Python interpreter. This library supports many file formats, and provides powerful image processing and graphics capabilities. It is used in the *Viewer* module only to generate the plot backgrounds and allow for updating them directly when the g range that is used and the maximum frequency to be shown on the FFT plot change.

- pylab: Also called matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. It is used in the *Report Generator* module only to generate the plots used in the reports.

- reportlab: This library is open source and is one of the most full-featured libraries for PDF creation available. It is used in the *Report Generator* module for generating reports in PDF format.

6.4 Modules

The python code implementing the application is divided into python modules (.py files) and python packages (folders containing .py files along with an __init__.py file). The following is a description of each one of those modules (or packages).

6.4.1 Reader

This module encapsulates classes representing three handler types as well as a thread which performs the reading and the sending of values to the sensor managers. It is implemented in C to improve performance.
Uses: Threading, bluetooth, serial

Class: BaseHandler

Represents a base class for the handlers to inherit from. The inheritance is on attributes only. These attributes are:

- The x, y and z arrays.
- The raw data array.
- The socket object.
- The g value associated with the sensor the handler represents.
- The handler's sensor manager.
- The length of the values unpacked.

Class: FileSimulation

An object of this class allows for the creation of a simulated socket that can be used by the file handler. Upon its creation, it is passed a file from which data is read and placed in three separate lists: one for the x values, one for the y values and one for the z values. In order to give it the same interface as a bluetooth socket, the following methods are implemented:

- fileno: just returns the sensor number the file is associated with.
- send: writes the character being sent to an input buffer.
- recv: creates a packet in the format Byte (FF), Byte (FF), x value, y value, z value, Byte (F0) which is sent back to the calling function. In addition to creating the packet, and in order to simulate the receiving from a bluetooth socket, a sleep is added every *file_sleep_frequency* times for the length of *file_sleep_time*. This will make it look closer to a sensor read, where data is not always available to be read.

Class: FileHandler

This class is used when wanting to simulate running data from a predetermined file. It is a handler whose socket is of type FileSimulation described above.

Class: SerialSimulation

Similar to the FileSimulation class, this class allows for the creation of a simulated socket object that can be used by the serial bluetooth handler. It inherits from the Serial class provided by the *serial* library, and is used when acquiring data on the Mac OS since the *bluetooth* library is not available for it. Therefore, its *send* method simply calls the serial *write* method, and its *recv* method simply calls the serial *read* method with the required characters passed.

Class: SerialBluetoothHandler

This class is similar to BluetoothHandler described below, except that its socket is of type SerialSumulaion rather than BluetoothSocket.

Class: BluetoothHandler

This class is used when acquiring data from the bluetooth, its socket is of type BluetoothSocket. The following are the most important methods of this class:

- setupBlueTooth: sets up the bluetooth connection so that the data reading can start. The setup process involves the following:
  1. Reading the "CONNECT" message (20 bytes).
  2. Sending "+++\r" and reading back 10 bytes.
  3. Sending "ATMF\r" and reading back 6 bytes.
- stop_data: stops reading by sending the '}' character, then waits for the stop message to be sent back to ensure that the reading has stopped. The stop message consists of 4 "FF" bytes in a row.

- stop_reader: stops the communication by closing the socket.

- getBatteryLevel: The battery level is read by sending '<' to the socket. The value read is then converted so that it appears as a percentage value. The minimum battery level allowed (equivalent to a 0%) is when the power drops by 2V from the maximum which is 6V. Therefore, the conversion is done as follows:

  1. Computing the A/D convertor's conversion factor which is given by: $conversion = 3.3 \times 1000/256\ mv/bit$ since it is an 8 bit A/D conversion and the input source is 3.3 V.
  2. Computing the bit value of the offset (2V) allowed for the voltage to drop by. This is given by: $offset = 2.0/conversion$.
  3. The bit value of the maximum voltage is 256 (12 bit A/D convertor). Therefore, the final battery percentage is: $100 \times (value - offset)/256$, where value is the second byte read back from the socket.

- getTemperature: The temperature is read by sending '!' to the socket. The value read is then converted so that it appears as a ${}^0C$ value. The conversion steps are as follows:

  1. Computing the A/D convertor's conversion factor which is given by: $conversion = 3.3 \times 1000/256\ mv/bit$ since it is an 8 bit A/D conversion and the input source is 3.3 V.
  2. Computing the bit value of the offset (the sensor reads 424 mV at 0 Celsius). This is given by: $offset = 424/conversion$.
  3. Computing the bit value of the slope (this comes from the temperature sensor's datasheet). This is given by: $slope = 6.25/conversion$.
  4. The final temperature value is: $(value - offset)/slope$, where value is the second byte read back from the socket.

- getCalibratiobParameters: This method should read the sensor's calibration parameters based on the given g value from the PIC attached to the sensor. The values are currently hardcoded and have been computed manually as described in 3.6. In the future, these values should come from the acquisition unit and should be computed for each unit using a test rig for different g values and different temperature ranges.

- setGValue: sets the g value on the sensor to the requested g value by sending '!' followed by a byte representing the g value according to the following mapping: 2.5:0xa8, 3.3:0xa9, 6.7:0xaa, 10:0xab where the value on the left is the g value and the value on the right is its byte representation on the sensor. Finally, it reads back the echoed byte.

Class: ReaderThread

This class represents a thread that is spawned when the user presses the "Run" button. The thread is killed when the user presses the "Stop" button. Upon its creation, it is passed the handler objects whose sockets it will be reading data from, and it creates a dictionary (SocketsDict) mapping the socket ID's to the handlers. While the thread is alive it performs the following:

1. Opens a connection with each one of the sockets (by sending the '{' character) and read the byte echoed back to make sure it is '}' (this indicates a successful connection).

2. Calls the appropriate read method (fileRead, blueToothRead or serialBluetoothRead according to the handler type). Each one of these methods will keep calling in a while loop the method *socketRead* explained below with the appropriate parameters. The while loop terminates when the thread is requested to be aborted via its *abort* method.

Method: socketRead

- Inputs: The socket ID, a boolean value indicating whether it is a file or a bluetooth socket, and the number of bytes to read from the socket.
- Description: Reads the required number of data bytes from the socket, calls the appropriate function to unpack the data (see the *Unpacker* module), sends the data to the Sensor Manager through its *processData* method only in multiples of four packets (the bandpass filter requires that).

Method: safeRead

This method takes a socket and a number of bytes n and reads from the socket exactly n bytes and finally returns the concatenation of those bytes.

6.4.2 Sensor Manager

Uses: numpy

Class: SensorManager

This module is implemented in C, as it is responsible for cycling through the acquired data in order to apply filters on it, log it, perform FFT on it, and store it in order for the viewer to access it. Its methods are:

Method: startUp

This method is called until there is fft_size data (this is determined by the settings menu FFT size option) in order to perform an initial FFT on the data to obtain the centre frequencies for the bandpass filters.

Method: processData

- Inputs: Three arrays of data (x,y, and z values) of the same length, and their length

- Description: Performs the following:

    1. Calls the *startUp* method until it is not required.
    2. Calls the *frequencyCheck* method.
    3. If it is a data acquisition (handler is bluetooth), calibrates and dc filters incoming data one axis at a time.
    4. If it is a file simulation, only dc filters the incoming data one axis at a time.
    5. If it is a data acquisition, logs the calibrated, dc filtered data one axis at a time.
    6. Stores the data retrieved from the above steps in a circularqueue data structure named raw_storage.
    7. Calls the *applyFilters* method to apply the required filters on the data.
    8. Finally, if enough data is received to apply the FFT computation (this is dependant on the fft_size parameter which can be set through the settings menu - FFT size), then the *performFFT* method is called.

Method: frequencyCheck

This method is responsible for checking when the centre frequency used for the band-pass filter changes requiring the filter to be updated in order to obtain new coefficients. The check relies on a frequency tolerance value which can be set in the settings menu using the Frequency Tolerance command.

Method: applyFilters

This method is responsible for applying the filters on the raw data producing the filtered data which is then stored in a circularqueue data structure named filtered_storage.

Method: performFFT

This method is responsible for requesting FFT computations on raw data in order to obtain the and store the following:

1. The RPM values for all three axes.
2. The frequencies and their absolute values which are later used by the *Viewer* module for plotting the FFT plots.
3. The fundamental frequency for each axis, used as the centre frequency for the bandpass filters.

This method also checks whether the system has stabilized or not allowing the user to start recording the data. The condition for stability is dependant on an RPM tolerance value which can be set in the settings menu using the RPM tolerance command.

Method: updateFilter

This method is responsible for requesting an update on the bandpass filter by passing it the current centre frequency for the appropriate axis, as well as the current bandwidth value which is set by the user.

Method: timeoutError

This method is responsible for notifying the viewer that the bluetooth handler associated with this sensorManager object has timed out, causing it to display an error message to the user.

Other methods exist representing setters and getters for the module's attributes.

6.4.3 Viewer

Since two applications have been developed (to run on a PDA and on a PC), and in order to make the code as reusable as possible, all of the modules except for

the *Viewer* module have been kept common between the two applications. The *Viewer* module contains implementation related to the graphical user interface which is the only aspect that differs between the two applications. Therefore, two modules named *PC_viewer* and *PDA_viewer* have been implemented, both inheriting from a module named *base_viewer* which includes the classes and methods that are common between the two applications. The following are the classes that each viewer module is composed of:

Class: LanguageSelectionPanel

Provides a panel for selecting the language that is to be used.

Class: ModeSelectionPanel

Provides a panel for selecting the mode to start the application with: File Simulation or Data Acquisition.

Class: InfoGatheringPanel

Provides a panel for the user to enter information pertaining to the data acquisition.

Class: DeviceDiscoveryPanel

Provides a panel that displays the discovered bluetooth devices, allowing the user to select the ones they would like to connect to, and assign them locations and orientations.

Class: MainFrame

Two kinds of main frames exist: a Main File Frame (created in File Simulation mode) which allows for the opening of files, and a Main Sensor Frame (created in Data Acquisition mode) which allows the connection to sensors and the recording of data.

Class: AcceptanceLevelsPanel

A panel that allows for the setting of the acceptance levels used in the tuning report generation.

Class: BearingsPanel

A panel that allows for adding new bearings and their diameters which are then used in the tuning report generation.

Class: InfoPanel

A panel that is part of the main frame, and consists on a common info panel, and a sensors info panel as described below.

Class: CommonInfoPanel

A panel that provides the acquisition information (customer, data, etc), as well as buttons for connection, starting, stopping, starting recording, and stopping recording.

Class: SensorsInfoPanel

A panel that provides information about the sensors (their locations, battery and temperature readings), as well as buttons for generating the reports (in data acquisition mode) and exiting and restarting the application.

Class: FFTPanel

A panel that encapsulates the FFT plots.

Class: WaveformPanel

A panel the encapsulated the waveform plots.

Class: LargeOrbitPanel

A panel that encapsulated the large orbit plot.

Class: SmallOrbitPlotsPanel

A panel that encapsulated the small orbit plots.

Class: ValuesPanel

A panel displaying the numerical values associated with each of the active sensors.

Class: OptionsPanel

A panel encapsulating all the options available to the user:

1. Setting on the g value.
2. Choosing the orientation to plot the orbit plot in.
3. Choosing the source of the centre frequency (manual or from the FFT).
4. Choosing the unit to display the stroke in (mm or inches).
5. A slider for setting the centre frequency. The maximum allowed frequency is 100 (this is caused by a 6000 RPM machine).
6. A slider for setting the bandwidth used for the filter. The maximum allowed bandwidth is 20 Hz.
7. A slider for setting the maximum frequency to be displayed on the x axis of the FFT plots. The maximum value is half the sampling frequency which is 500Hz.

Class: ReportingPanel

This class exists only for the PC application. It allows for a window to show up when the user chooses to generate a tuning report. This window contains various parameters that the user is allowed to enter or adjust. The user can preview the values generated based on these parameters before the PDF report is actually generated. When they are satisfied with the results after tuning the parameters as required, they can choose to generate the report.

6.4.4 Logger

Class: Logger

An object of type Logger is responsible for logging the data it receives. Such an object is instantiated for each location being monitored (i.e for each sensor manager). Each object has its own *log* and *stop_logging* methods. The *log* method can be modified in order to allow for a different logging scheme.

6.4.5 Calibrator

Class: Calibration

This class is part of the *scaling* package and is implemented in C to increase performance. Upon creating an object of this type, it should be passed the slopes and the offsets used to create the lines that are used to calibrate the data in all three axes. When its *scale* method is called with the x, y, and z values that are to be scaled, the calibrated data is computed as follows for each axes:

$$x = x_slope * x + x_offset \tag{6.1}$$

Finally, the calibrated x, y, and z values are returned.

6.4.6 Filter

The Filter module is implemented as a python package which includes sub-modules each representing a filter type. Filters created in the future should be added to this package. Each filter should be implemented as a class with a method named *calc* performing the filter's calculation. The *calc* method accepts a list of samples as an input and returns a list of those sample, but filtered. Other methods providing services to the *calc* method can be implemented as part of each filter class.

Class: AllPass

The Allpass filter is applied to the z-axis values (perpendicular to the machine's main movement) only. These values do not represent the machine's main movement and are therefore not that important to filter. This class's *calc* method simply returns the values passed to the filter untouched.

Class: Butterworth

The Butterworth filter is the bandpass filter used to produce the filtered data used throughout the application. This module is implemented in C in order to improve performance, as the filtering occurs on every data sample read from the sensror. Its *calc* method implements an IIR filter used to filter a list of samples. the IIR filter computation is implemented in such a way that allows for an increase in performance, namely by eliminating the need to re-compute values through the reuse of data. For this reason, the filter is to be applied on arrays of samples whose length is a multiple of four. The following equations explain how this is done on every 4 samples (example based on a 2nd order filter):

$$result_1 = y(n) = a_1x(n) + a_2x(n-1) + b_1y(n-1) + b_2y(n-2) \tag{6.2}$$

$$result_2 = y(n+1) = a_1x(n+1) + a_2x(n) + b_1result_1 + b_2y(n-1) \tag{6.3}$$

$$result_3 = y(n+2) = a_1x(n+2) + a_2x(n+1) + b_1result_2 + b_2result_1 \tag{6.4}$$

$$result_4 = y(n+3) = a_1x(n+3) + a_2x(n+2) + b_1result_3 + b_2result_2 \tag{6.5}$$

Once the filter is created and every time its bandwidth or centre frequency change (when its *update* method is called) the filter coefficients need to be computed. Other helper methods exist as part of this class, and are mostly based on the butterworth filter code provided by http://www.exstrom.com/journal/sigproc/index.html.

Class: DC

This class is implemented in C to increase performance. Its *calc* method applies a dc filter to the incoming sample x as follows:

$$y(n) = x(n) - x(n-1) + R * y(n-1) \tag{6.6}$$

where R is a constant value between 0.9 and 1, and depends on the sampling rate and the low frequency point.

6.4.7 FFT Computer

Uses: numpy

Method: getFFT

- Inputs: Sequence of data to perform FFT on, sampling rate.

- Description: Peforms the following:

    1. Applies an FFT computation on the given data to obtain the frequencies and their absolute values.
    2. Calls polynomialInterpolation on the three values representing the absolute values of the frequenxy with the highest absolute value and the two frequencies surrounding it.

3. Computes the fundamental frequency as follows:

$$fund_freq = (max_index - 2 + interpolant) * correction \tag{6.7}$$

where:
max_index is the index of the frequency with the maximum absolute value
interpolant is the result of polynomialInterpolation
correction = samplerate/data_length

4. Computes the RPM as follows:

$$rpm = fund_freq * 60 \tag{6.8}$$

- Outputs: RPM, frequencies, frequency gains, fundamental frequency.

Method: polynomialinterpolation

- Inputs: 3 points to perform interpolation on

- Description: Peforms the following:

1. Given the three points, performs a polynomial interpolation on the polynomial given by:

$$ax^2 + bx + c = 0 \tag{6.9}$$

by solving for the point where the slope is zero (this will give the frequency with the highest absolute value):

$$2ax + b = 0 \tag{6.10}$$

or equivalently,

$$x = -b/2a \tag{6.11}$$

The point x then is the point at the peak of the polynomial curve where the slope is zero.

- Outputs: interpolant.

6.4.8 Numerical Values Computer

Method: getGForces

- Inputs: three lists of data (x_f, y_f, and z_f) representing the filtered values in the x, y, and z axes.

- Description: Computes the g forces as follows:

  1. Finds the minimum value for each axis.
  2. Finds the maximum value for each axis.
  3. Averages the minimum and maximum values for each axis.
  4. Computes the main g force as follows:

$$Main\ g\ force = max\{\sqrt{(x^2 + y^2)}, for\ x \in x_f\ and\ y \in y_f\} \quad (6.12)$$

- Outputs: Average g forces for x, y, z axes, and main g force.

Method: getStrokes

- Inputs: RPM, Average g forces for x, y, and z axes, and main g force.

- Description: Computes the strokes as follows:

  1. For each axis, the stroke (in mm) is calculated as follows:

$$Stroke = \frac{axis\ g\ force}{1000 * 2 * (RPM/60)^2} \quad (6.13)$$

  2. Computes the main stroke as follows:

$$Main\ stroke = \frac{Main\ g\ force}{1000 * 2 * (RPM/60)^2} \quad (6.14)$$

- Outputs: Strokes for x, y, z axes, and main stroke.

6.4.9 Ellipse Fitter

Uses: numpy Note: The implementation of the ellipse fitting method is based on the Matlab ellipse fitting module. Additionally, at least five points are required to perform the ellipse fitting on, the more points passed to the function, the better fitted will the ellipse be.

Method: getEllipseParameters

- Inputs: Lists of filtered data in the x and y axis which determine the machine's main movement.
- Description: Performs the following:
  1. Finds the estimate of the conic equation representing the ellipse which is given by:
  $$ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{6.15}$$
  2. Using the given points, solves the equation using the linear least squares method. The *lstsq* function provided by *numpy.linalg* is used here.
  3. Extracts the parameters (a,b,c,d,e) from the conic equation.
  4. Checks that the conic equation represents an ellipse. This is done by checking if $a \times c > 0$.
  5. Calculates the ellipse's major and minor axes lengths. This is done as follows:
  $$F = 1 + (d^2)/(4 * a) + (e^2)/(4 * c) \tag{6.16}$$
  $$a = \sqrt{(F/a)} \tag{6.17}$$
  $$b = \sqrt{(F/c)} \tag{6.18}$$
  $$major_axis = 2 * max(a, b) \tag{6.19}$$
  $$minor_axis = 2 * min(a, b) \tag{6.20}$$

6. Computes the ellipse's foci as follows:

$$foci = \sqrt{((major_axis/2.0)^2) - ((minor_axis/2.0)^2)} \tag{6.21}$$

7. Computes the ellipse's eccentricity as follows:

$$eccentricity = (2 \times foci)/major_axis \tag{6.22}$$

8. Computes the ellipse's phase angle as follows:

$$phase\ angle = 0.5 \times arctan\frac{1}{(foci - (major_axis/2.0))/minor_axis} \tag{6.23}$$

- Outputs: Eccentricity, Phase angle.

6.4.10 Report Generator

Uses: pylab, reportlab, numpy

Class: SinglePointReport

The methods of this class are responsible for creating a PDF report for a single monitored location on the machine using the data passed to an object of this type open its creation. Please refer to Appendix C for an illustration of the resulting single point report.

Class: OrbitReport

The methods of this class are responsible for creating a PDF report acting as a summary of the orbits produced by all the monitored locations. The report is created using the data passed to an object of this type open its creation. Please refer to Appendix C for an illustration of the resulting orbit summary report.

Class: TuningReport

The methods of this class are responsible for creating a PDF report providing recommendations for adjusting the machine. The report is created using the data passed to an object of this type open its creation. The calculations performed here are all based on the old reports produced by the company. Please refer to Appendix C for an illustration of the resulting tuning report.

6.4.11 Device Discovery

Uses: threading

Class: DiscoveryThread

This class is implemented as a thread and is responsible for discovering the available bluetooth devices within range. It returns only those ones whose names start with *SparkFun-BT* representing the bluetooth devices in each of the data acquisition units.

6.4.12 Unpacker

This module is a helper module for the *Reader*, and is implemented in C as its methods are called frequently. Its responsibility is in unpacking the data read from the handlers (bluetooth or file) according to a predefined format they are expected to be in and based on the sensor's orientation (i.e. whether the sensor is mounted to the top or the side of the machine). If the sensor is mounted to the top, simply the y and z values read from the sensor are swaped. This module's methods are:

Method: get_values_for_file_simulation

This method is responsible for unpacking data read from a file handler.

- Inputs: The handler's raw data array, its length, the orientation of the sensor associated with that hander, the handler's x,y, and z arrays and a pointer to a num_values integer that represents the number of values that were unpacked.

- Description: Unpacks the raw data based on the given format. Note that file simulation uses calibrated float values so the format is : Byte,Byte, float, float, float, Byte. Also note that the file handler will put data in this format in order to make the unpack operation similar to that of the bluetooth, allowing for the reuse of code.

- Outputs: Returns a pointer to the last point unpacked in the raw data array for the next time this function is called.

Method: get_values

This method is responsible for unpacking data read from the bluetooth handler. It shares the same interface as the *get_values_for_file_simulation*, but calls the *my_unpack* method to perform the unpacking on the values.

Method: my_unpack

This method unpacks data based on the 9 byte packet format used for the bluetooth handlers. The format is as follows: Byte (FF), Byte (FF), Short - 2 Bytes(x value), Short - 2 Bytes (y value), Short - 2 Bytes (Z value), Byte(F0).

6.4.13 Screen Scaling

This class is part of the *scaling* package and is implemented in C to increase performance. Its methods deal with scaling the data so that it appears correctly on the plots displayed on the screen. It is therefore dependant on the display size. It contains the following methods:

- 4 methods for scaling the data on the orbit plot. Each one is related to a different g range, and based on the g value passed to *orbit_scale* the appropriate method is used.

- 4 methods for scaling the data on the waveform plots. Each one is related to a different g range, and based on the g value passed to *waveform_scale* the

appropriate method is used.

- A method for scaling the data on the x axis of the FFT plots. This data represents the frequencies.

- A method for scaling the data on the y axis of the FFT plots. This data represents the gains. Note that all gains plotted are a ratio of that gain to the maximum gain, meaning that the maximum gain appears as '1' on the y axis.

6.4.14 Circular Queue

This is a helper module for the *SensorManager* module, providing it with a data structure to store m unfiltered and filtered data, where m is the size of the queue. The main feature of this data structure is that it will always return the most recent n data, where n is the requested data size. Since data is appended as soon as it is read, then data is being put in the queue as a rate of 1000 samples/ sec. Data is then pulled out of the queue when the viewer is ready to view it. In order to ensure that all the data stored is viewed, data should be pulled at a faster rate than that it is being put in. Figure 6.1 illustrates how the queue works. The Circular Queue module is implemented in C to improve performance, and consists of the following class:

Class: CircularQueue

This class represents the circular queue data structure described above. Upon its creation, it is passed a *size* parameter defining the maximum data size it can hold at once. It internally keeps track of a *start* and *end* pointer to indicate the oldest and the newest data. Its methods are:

- append : This method received 3 data points (x, y, and z) and append them as a tuple at the end of the queue, incrementing the *start* and the *end* pointers accordingly.

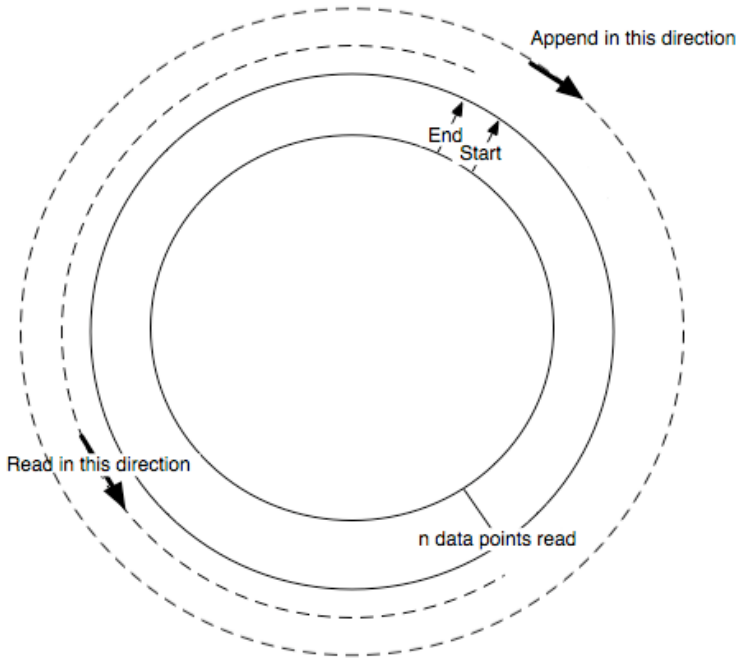


Figure 6.1: An illustration of the circular queue data structure.

- get : This method receives a length n of data points to send back. It creates a *getting_pos* pointer which will point to the last data point to read which is n points away from the end. It sends back three separate lists of data for each one of the axes. The data sent back is always the most recent n data points.

Chapter 7

Software Testing

7.1 Overview

Software verification is the final step prior to the delivery of the software. Verification of a software product involves all the activities that are undertaken to ascertain that the software meets its objectives [7]. One such activity is *testing*. Testing involves developing test cases or scenarios with specific inputs that the program is then run with in order to gain confidence about the quality of the software. The approaches taken in order to test the vibration analysis tool developed here are discussed in the remainder of this chapter.

7.2 Unit Testing

Experimental data from industrial projects have shown that the cost of removing an error after the software has been developed completely is much higher than if errors are eliminated earlier [7]. This is where unit testing proves to be very useful.

Unit testing, often referred to as *Testing in the small*, addresses the testing of individual modules. When testing each module, two types of testing schemes are conducted: White-box testing, and Black-box testing.

White-box testing (also referred to as structural testing) uses the internal structure

of the program to derive test data. It tests what each module is actually doing. For example, white-box tests may include testing *loops* to make sure they are finite, testing *if* statements to make sure that all possible conditions are covered, testing *try-except* statements to make sure that appropriate exceptions are caught, and so on.

Black-box testing (also referred to as functional testing) does not deal with the internal structure of the module. It tests what the module is supposed to do, meaning it is based on the program's specification rather than its structure. A method of applying black-box testing is by categorizing input data that is then passed to the module and comparing the observed results against the expected ones.

As in many of the vibration analysis tool modules, modules sometimes use other modules, and therefore testing them requires information from the modules they use. For example, if a module uses an operation that is part of another module, this operation needs to be simulated in order to test the original module. In cases like this, *stubs* and *drivers* are used.

Stubs are used to simulate a procedure with the same inputs and outputs as the missing procedure, but with a simplified behaviour. For example, values can be hard coded in the stub, or they can be read from a file or asked to be inputted by the user.

Drivers on the other hand simulate the use of the module being tested. The driver sets the values of the data as it would be set in the real application by the missing modules.

In general, unit testing allows for verifying the correctness of the system's functions. In order to verify the system's software qualities, system testing is performed.

7.3 System Testing

System testing involves testing the system as a whole and is often referred to as *Testing in the large*. When performing system testing, the method of separation of concerns should be employed in order to design test cases that will test various aspects of the system reflecting the system's requirements document. It is important to verify

all the software qualities, meaning that not only should testing address compliance with the functional requirements, but also with non-functional requirements such as performance, robustness, and usability. The following is a discussion of some software qualities and how they were tested:

- Performance: In order to test for performance, the system was tested under peak conditions. The most important tests here are:

    1. Testing the system when communicating with eight sensors at once (which is the maximum number of sensors the system can communicate with).
    2. Testing how well the bluetooth receiver performs as its distance from the transmitter on the acquisition device increases.
    3. Testing how well the system performs at the minimum and maximum temperatures the sensor can operate at. (This test was not conducted, but is recommended for the future).
    4. Testing how well the system performs at the point where the battery level approaches zero.

- Robustness: In order to test for robustness, the system was tested under unexpected conditions such as:

    1. Erroneous user commands. This could happen when setting values through the settings menu, or when assigning sensors to locations on the machine. The viewer module is responsible for checking the data entered and notifying the user of any errors.
    2. Erroneous data coming from the sensor units. The reader module is responsible for checking this by parsing the data and making sure it complies with the packet structure it should be in.
    3. Loss of connection with the sensor units due to a bluetooth error or disconnection. The system is responsible for detecting this and notifying the user while allowing the rest of the system to continue running uninterrupted.

- Portability: This was tested by running the system on the platforms stated in the requirements document and ensuring it operates as expected The platforms that the system was tested on are Mac OS, Windows XP and the version of Linux running on the PDA.

- Usability: This was tested by having regular users of the software use the system and obtain feedback from them regarding the usability of the system. Both engineers and technicians who would be using the software regularly were interviewed and their input on the usability of the developed software application was taken into consideration.

Finally, both in-lab and on-the-field tests were conducted in order to test the software system as a whole. In the lab, a calibration unit vibrating at a known frequency was utilized in order to test the system on a small scale. This unit helped in verifying that each of the sensor units was working properly. On the field, multiple sensors were attached to different types of machines that the company produces and data was acquired. After that, reports were generated and their results were compared to results produced using the company's current solution. This is also referred to as *Parallel Testing*. Additionally, we were provided with a large amount of data files containing vibration data that the company had acquired with their previous system. These files were played back using the "File Simulation" mode in our tool and their numerical results (such as resulting RPM, stokes, g values, etc.) were compared to the results generated by our tool which helped increase our confidence level in it.

Chapter 8

Error Analysis

8.1 Overview

This chapter discusses the various sources that can cause an error in the readings obtained by the data acquisition devices or in the computations performed in the vibration analysis application.

8.2 Types of Errors

Since the units that are used for data acquisition are composed of more than one type of electronic device, and each device has its own resolution, the constructed acquisition device will have a margin of error on the values sent from it through bluetooth. The following is a list of sources of error caused by the different electronic parts used in the acquisition device:

1. Error on the source voltage caused by the change in temperature: The source voltage changes by $50 * 10^{-6}$ Volts for every degree Celsius change in the temperature.

2. The 12 bit A/D convertor has its own resolution given by:

$$A/D\ resolution = \frac{Source\ Voltage}{A/D\ range} = \frac{3.3V}{2^{12}} = \frac{3.3 * 1000\ mV}{4096} = 0.8057\ mv/bit \tag{8.1}$$

This value is fixed for a fixed source voltage of $3.3V$.

3. Electric error causing around 3 bits of inaccuracy on the A/D convertor readings: This is noticed when monitoring a stable readings and noticing that its values fluctuate around the reading within a 3 bit range.

4. The sensor has a different resolution range for each one of the g levels that it can operate on. Table 4 displays the 4 ranges and their corresponding resolutions.

G Range	Resolution
2.5 g	480 ± 36 mV/g
3.3 g	360 ± 27 mV/g
6.7 g	180 ± 13 mV/g
10.0 g	120 ± 9 mV/g

Table 8.1: G ranges and their resolutions

5. Temperature noise on the sensor: It is important to note that Table 4 represents the resolutions when the temperature is 25^0C and the source voltage is $3.3V$. However, any change in temperature will also add an error on the sensor readings, and this is given by $\pm 0.03\%/^0C$

6. The temperature sensor itself has an accuracy range between $\pm 2^0C$ and $\pm 3^oC$ when the temperature is around 25^oC. Please refer to the temperature sensor datasheet that can be found at *http://cache.national.com/ds/LM/LM60.pdf* for more information.

7. RMS noise on the sensor: Equivalent to an error of 4.7 $mVrms$. Refer to the sensor data-sheet.

8. Power Spectral Density RMS noise on the sensor: Equivalent to an error of 350 $\mu g/\sqrt{Hz}$. Refer to the sensor data-sheet.

The following are the sources of error that affect the results produced by the vibration analysis software:

1. FFT Resolution: This is equivalent to $\frac{sampling\ rate}{data\ length}$.
   As the length of the data passed to the FFT function increases, the resolution decreases giving a more accurate fundamental frequency and therefore RPM result. This resolution parameter is also refered to as the FFT *bucket* or *bin* size. It is important to keep in mind that the data length should be a power of two in order to perform the FFT computation.

2. Calibration: As discussed in Section 3.6, the method used to calibrate the values read from the sensor units will highly affect the accuracy of the results produced by the vibration analysis application.

3. Ellipse Fitting: The accuracy of the phase angle and the eccentricity values generated based on fitting the data to an ellipse, depends highly on the number of values the fitting is performed on. As the size of the data used increases, the more accurate the ellipse fitting result becomes, but the more expensive the computation will be.

Chapter 9

Conclusion

9.1 Discussion

The focus of this thesis work has been on the development of a vibration analysis tool that can be used specifically in the field of mining. We believe that the developed tool has fulfilled its specified requirements. Firstly, it communicates wirelessly with data acquisition units which pick up data through acceleration sensors. It can communicate with a maximum of eight units at once, providing synchronous readings as opposed to most data acquisition methods which require acquiring data from one point of the machine at a time. Secondly, signal processing is applied on the acquired data. The method for computing the centre frequency for the bandpass filters used to filter the noise from the signals, is determined through a high precision method. The method is based on performing an FFT computation on the data and thereafter performing an interpolation around the frequency with the highest amplitude, bringing the result as close as possible to the actual fundamental frequency of the machine. Of course, the accuracy of this method depends highly on the size of the data points passed to the FFT computation. It is also worth mentioning here, that the current version of the vibration analysis tool computes the fundamental frequencies based on the data acquired from *all* the used sensors. This ensures that each location's data is being filtered with a bandpass filter centred at the fundamental frequency coming from

that location. However, if the performance of the tool becomes of importance, and since an FFT computation along with the interpolation are expensive operations, the performance can be boosted by obtaining a fundamental frequency estimation from one sensor only. This estimation can be safely employed since the machine's fundamental frequency should be the same on all its parts. If a significant noise occurs on one of the parts causing its fundamental frequency to change, it will affect the remaining parts changing the overall fundamental frequency. Lastly, the vibration analysis tool developed provides reports that are used to tune and adjust the machines in order to increase their performance and lifetime.

We believe that this tool will prove to be very beneficial in the mining industry, especially due to its wireless communication method which is a fairly new method in this field. Soon, all vibration analysis tools used in the mining field will need to allow for wireless communication. The Mine Safety and Health Administration department at the U.S. Department of Labor has issued a new act stating the following: "The MINER Act requires that mine operators adopt wireless communications and electronic tracking systems by June 2009" (http://www.msha.gov). This is an indicator of the advancement level of the developed tool.

9.2 Future Work

The following is a list of future work that can be based on the developed vibration analysis tool, or added as an extension to improve its performance:

1. As presented briefly in Section 2.6.2, *Fast Wavelet Transform Analysis* can be applied in addition to the Fourier Analysis in order to better analyse the acquired data and detect unwanted noise.

2. As mentioned in Section 3.6, a more accurate method for calibrating the sensors can be employed. One such method would involve a test rig with a predefined set of g ranges that it can operate on, onto which the sensors can be attached in order to acquire data for calibration in all three axes. Additionally, the effect of

temperature on the readings should be taken into consideration, and calibration should be performed on various temperature ranges utilizing the temperature sensors that are a part of each acquisition device.

3. Researches interested in performing diagnostics on the acquired data are able to base their work on the developed tool. Diagnostics are then performed by creating a set of predefined mechanical problems that can occur on the machines, and comparing acquired data against them. This will prove to be very beneficial for the company in that it will help detect problems at an early stage which will help decrease machine down time and maintenance costs.

4. Researches interested in performing feedback control on the machines can base their work on the developed vibration analysis tool. A method of sending signals based on the acquired data back to the machine's motors would have to be developed and would ultimately allow for the control of the machine's behaviour.

Bibliography

[1] M. L. Adams, *Rotating Machinery Vibration: From Analysis to Troubleshooting.* CRC Press, 2001.

[2] Bluetooth "http://www.bluetooth.com", last visited June 7, 2008.

[3] L. D. Paarmann, *Design and Analysis of Analog Filters. A Signal Processing Perspective.* Springer, 2001.

[4] Freescale Semiconductor, *2.5g - 10g Three Axis Low-g Micromachined Accelerometer*, 2007. MMA7261QT.

[5] W. Gander, G. H. Golub, and R. Strebel, "Least-square fitting of circles and ellipses," *BIT*, vol. 43, pp. 558–578, 1994.

[6] D. Gerhard, "Pitch extraction and fundamental frequency: History and current techniques," Tech. Rep. TR-CS 2003-06, Department of Computer Science, University of Regina, 2003.

[7] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering.* Prentice Hall, 2002.

[8] L. Kim, *A Review of Rolling Element Bearing Health Monitoring.* Engine Laboratory, Division of Mechanical Engineering, National Research Council Canada, 1982.

[9] R. G. Lyons, *Understanding Digital Signal Processing.* Prentice Hall, 2004.

[10] T. W. Parks and C. S. Burrus, *Digital Filter Design*. New York: Wiley, 1987.

[11] P. L. Rosin, "Further five point fit ellipse fitting," in *British Machine Vision Conference*, 1999.

[12] I. W. Selesnick and C. S. Burrus, "Generalized digital butterworth filter design," *IEEE Transactions on Signal Processing, Vol. 46, No. 6*, June 1998.

[13] SparkFun "http://www.sparkfun.com", last visited June 7, 2008.

[14] G. Thomas, M. Simpson, R. D. Winton, and S. D. Robinson, "System analysis of rotors, supporting structures and their foundations," tech. rep., Structural Dynamics Research Corporation, 1982.

[15] K. Tuck, "Implementing auto-zero calibration technique for accelerometers," Tech. Rep. AN3447, Freescale Semiconductor, 2007.

Appendix A

User Manual

This appendix serves as a guide for the user on how to use the vibration analysis tool.

A.1 Start Up

Upon starting the application, you will be asked select their language of preference as in Figure A.1. Once the you have selected your language from the drop-down list, press the "Done" button.

Next, you are asked to select the mode you would like to start the application in as in Figure A.2. Choose "File Simulation" if you would like to play back previously acquired files, or choose "Data Acquisition" if you would like to connect to the available sensor units and acquire data.

A.2 File Simulation Mode

Upon choosing "File Simulation", the main file simulation window will appear as in Figure A.3.

Next, you can open files by accessing the "File" menu, and selecting either "Open Directory" or "Open Old File". The "Open Directory" option will allow you to choose a directory containing one or more files corresponding to one or more sensors

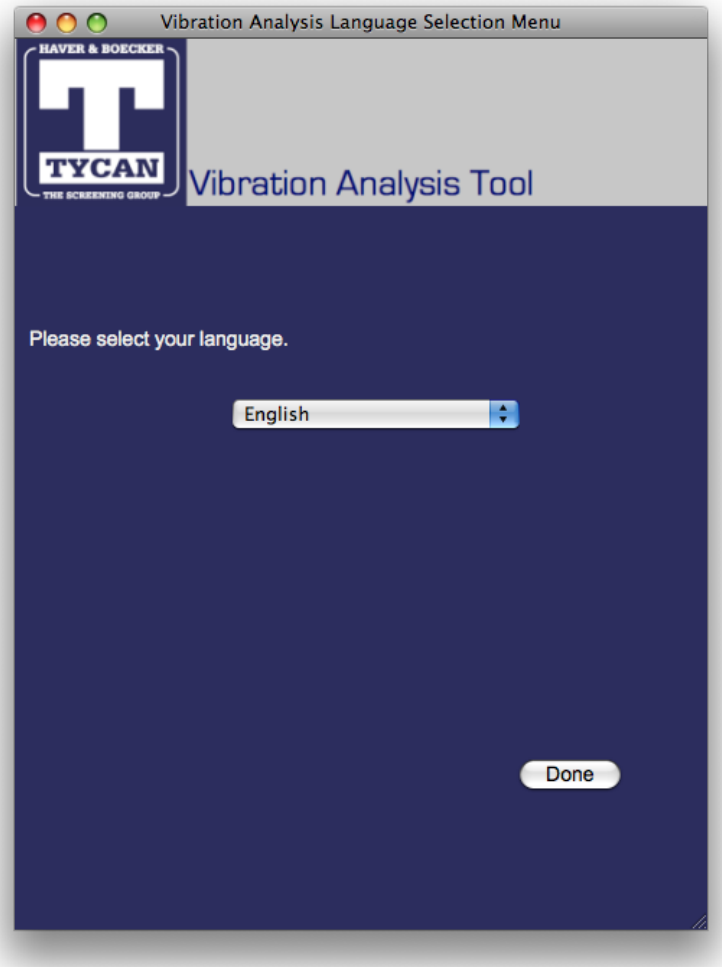


Figure A.1: Language Selection Menu.

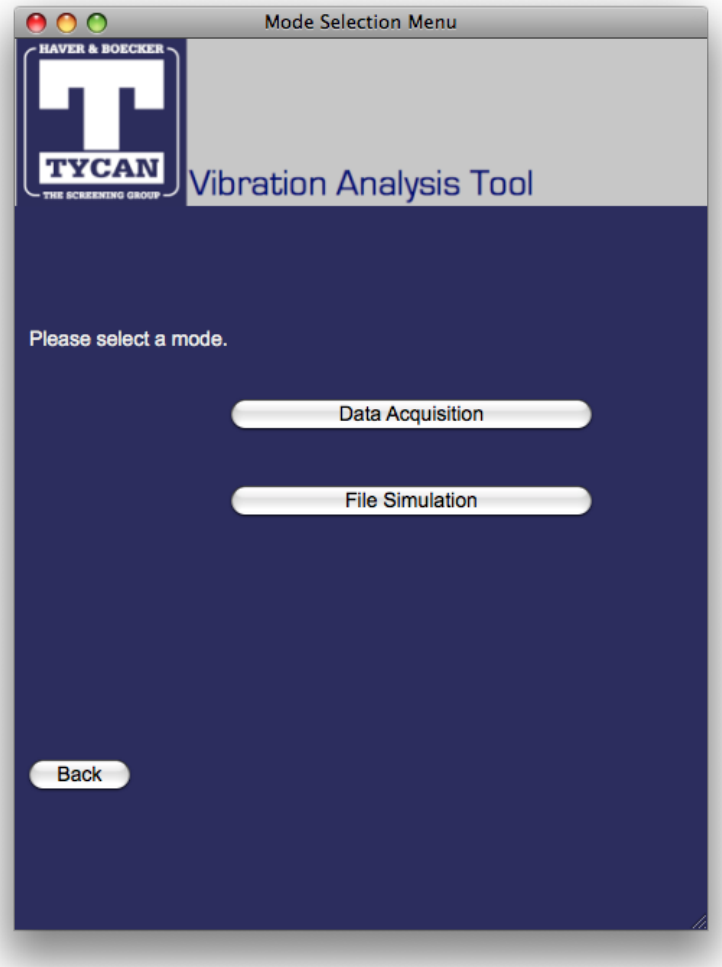


Figure A.2: Mode Selection Menu.

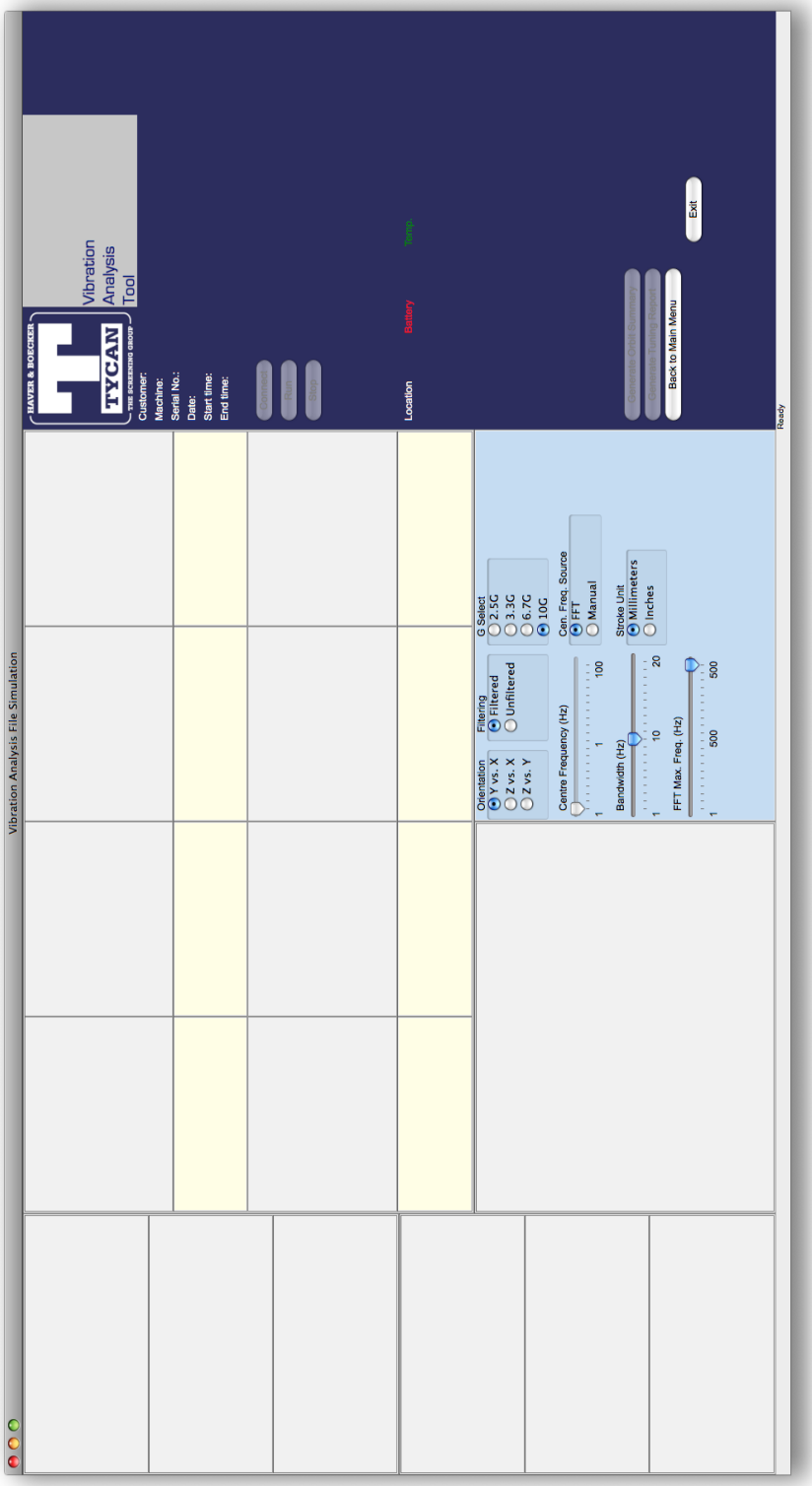


Figure A.3: Main view of File Simulation Mode..

used to acquire data during that acquisition. These folders are named with the date and time the data was acquired. On the other hand, the "Open Old File" option should be used when you would like to playback a file that was acquired using the old system. Note that in this case the location, battery and temperature values will not appear as they are not available in the old files, and report generation is disabled.

Upon opening a file or a directory, you will need to press the "Connect" button which will prepare to start simulating from the files. Upon connection (you will see "File ready" in the status bar"), press the "Run" button to start playing back the data as in Figure A.4. You can press the "Stop" button anytime to end the playback.

Additionally, while the playback is stopped, you can add additional directories to playback. This is particularly useful if data had to be acquired on two separate acquisitions due to a lack of functional sensor units for example, and the user would like to play back the data simultaneously since it was coming from the same machine. This can be done by accessing the "File" menu, and selecting the "Add Directory" option. You will then need to press the "Connect" button again before running the simulation.

A.3 Data Acquisition Mode

Upon choosing "Data Acquisition", you will be transferred to a window where you will enter information relating to that particular acquisition as in Figure A.5. While you are entering this information, the available bluetooth devices will be detected. Once the detection is complete the "Done" button will be enabled. Once it is enabled, press it to proceed to the next step.

The next window will display the detected bluetooth devices that are related to the vibration analysis tool. You will then need to assign each device a location from the list, as well as a top/side orientation based on where it is attached on the machine. You can uncheck the box beside each device if you do not want to connect to it. This is shown in Figure A.6.

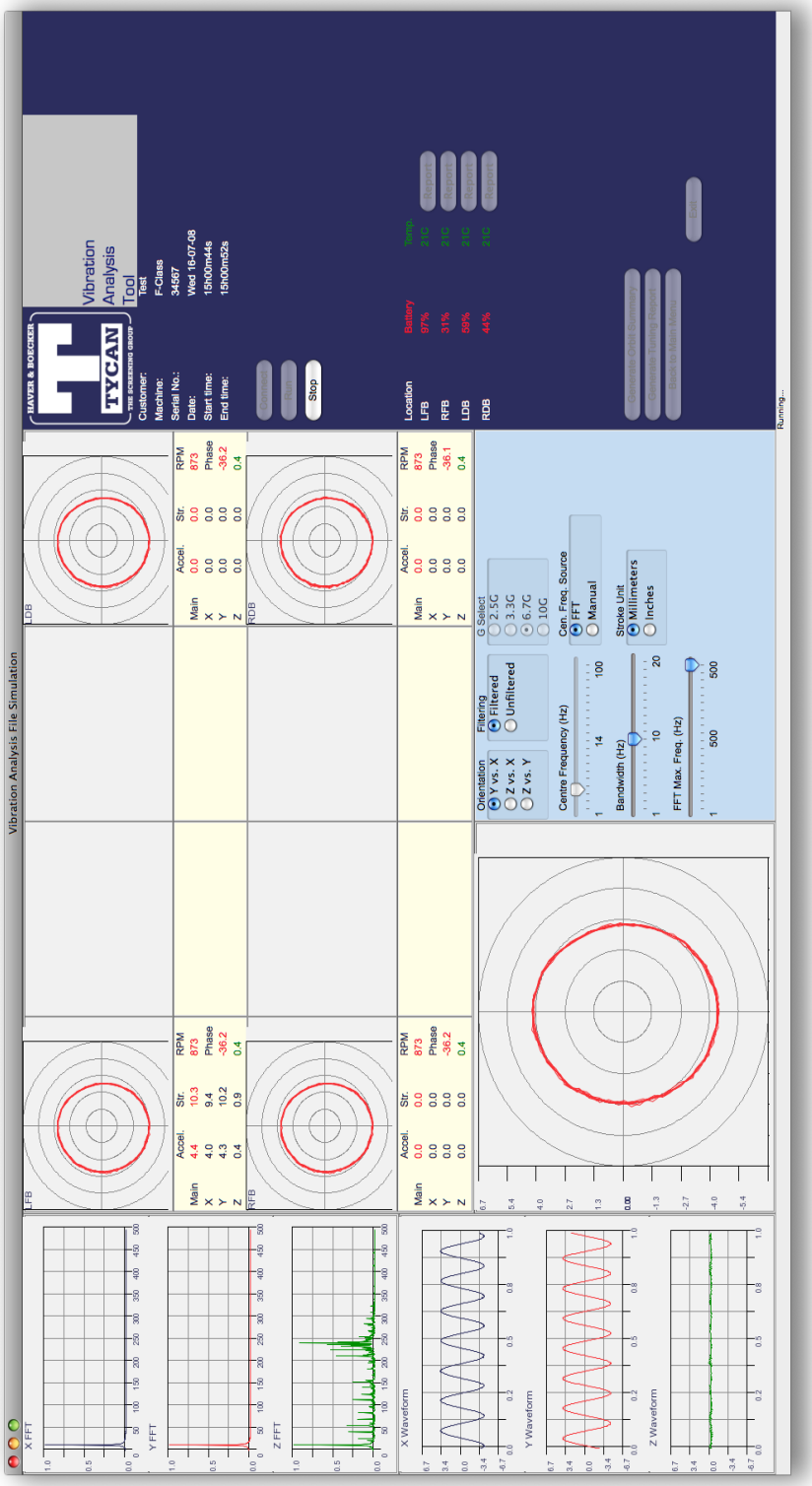


Figure A.4: Main view of File Simulation Mode while running.

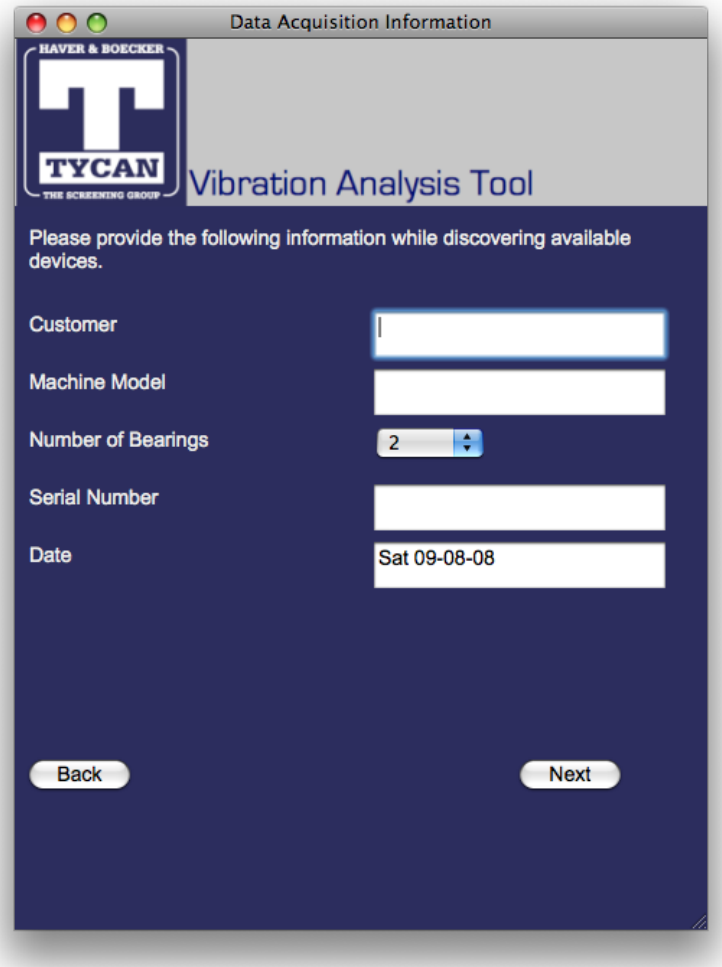


Figure A.5: Acquisition Information Gathering Menu.

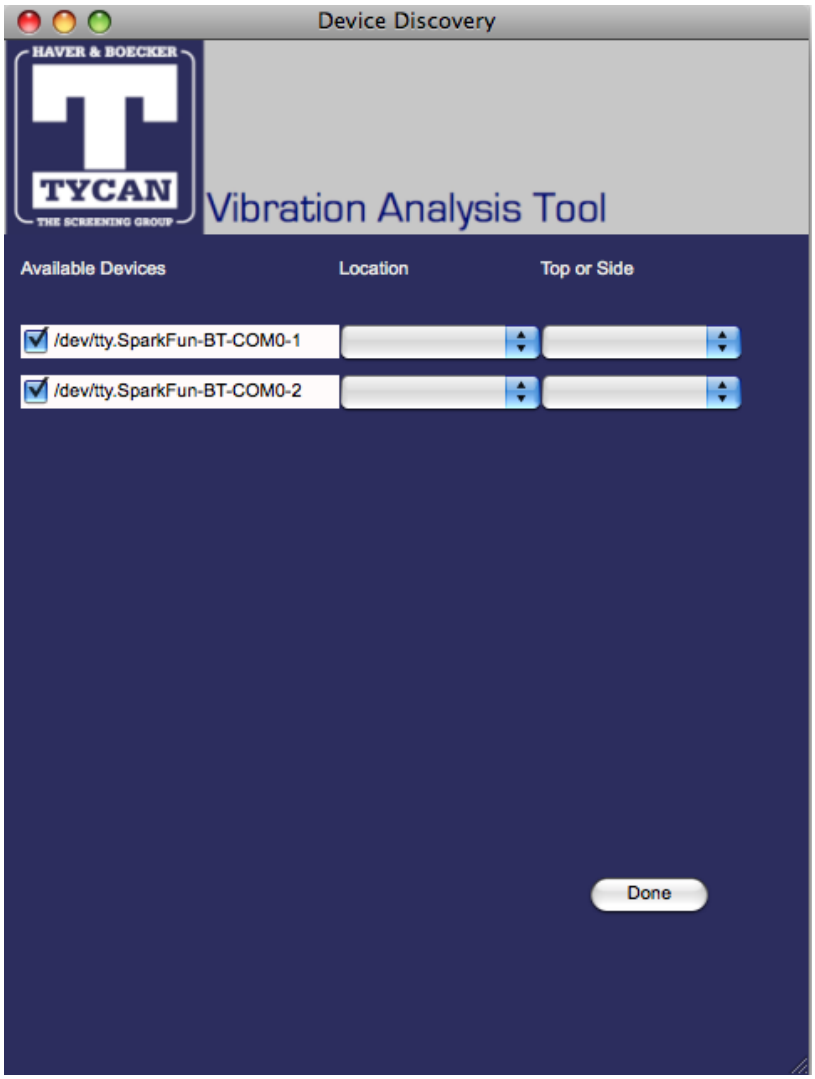


Figure A.6: Device Selection Menu.

Upon pressing the "Done" button, the main view of the data acquisition mode will appear, with the selected sensors appearing on the right as in Figure A.7.

Upon pressing the "Connect" button, the system will connect to the chosen bluetooth devices, and will obtain the battery and temperature readings from each device. Moreover, the g value chosen in the options panel will be sent to each of the devices so data can be acquired within that g range.

Next, you can press the "Run" button to start reading the data from the sensors. As soon as the system has stabilized, the "Start Recording" button will be enabled. Press it to start recording the data into files, and press the "Stop Recording" button to end the recording. Pressing the "Stop" button will stop reading the data from the sensors, and will update the battery and temperature values.

A.4 Settings

The following settings are available under the "Settings" menu in both modes:

1. Centre Frequency Tolerance: This is the tolerance on the centre frequency (measured in Hz) used to determine whether or not the bandpass filter used should be updated and new coefficients for it should be computed. Figure A.8 displays how this tolerance can be changed.

2. RPM Tolerance: This is the tolerance on the RPM (measured in RPM) used to determine whether or not the system has reached a stable point and recording of the data can be allowed. Figure A.9 displays how this tolerance can be changed.

3. Update Frequency: This parameter defines how frequently the plots are updated with the data. It is measured in seconds. As its value increases, the plots are updated slower. This parameter can be changed as in Figure A.10.

4. Update History: This parameter defines the size of data the be plotted on every update. As this size increases, more data is being plotted in the past.

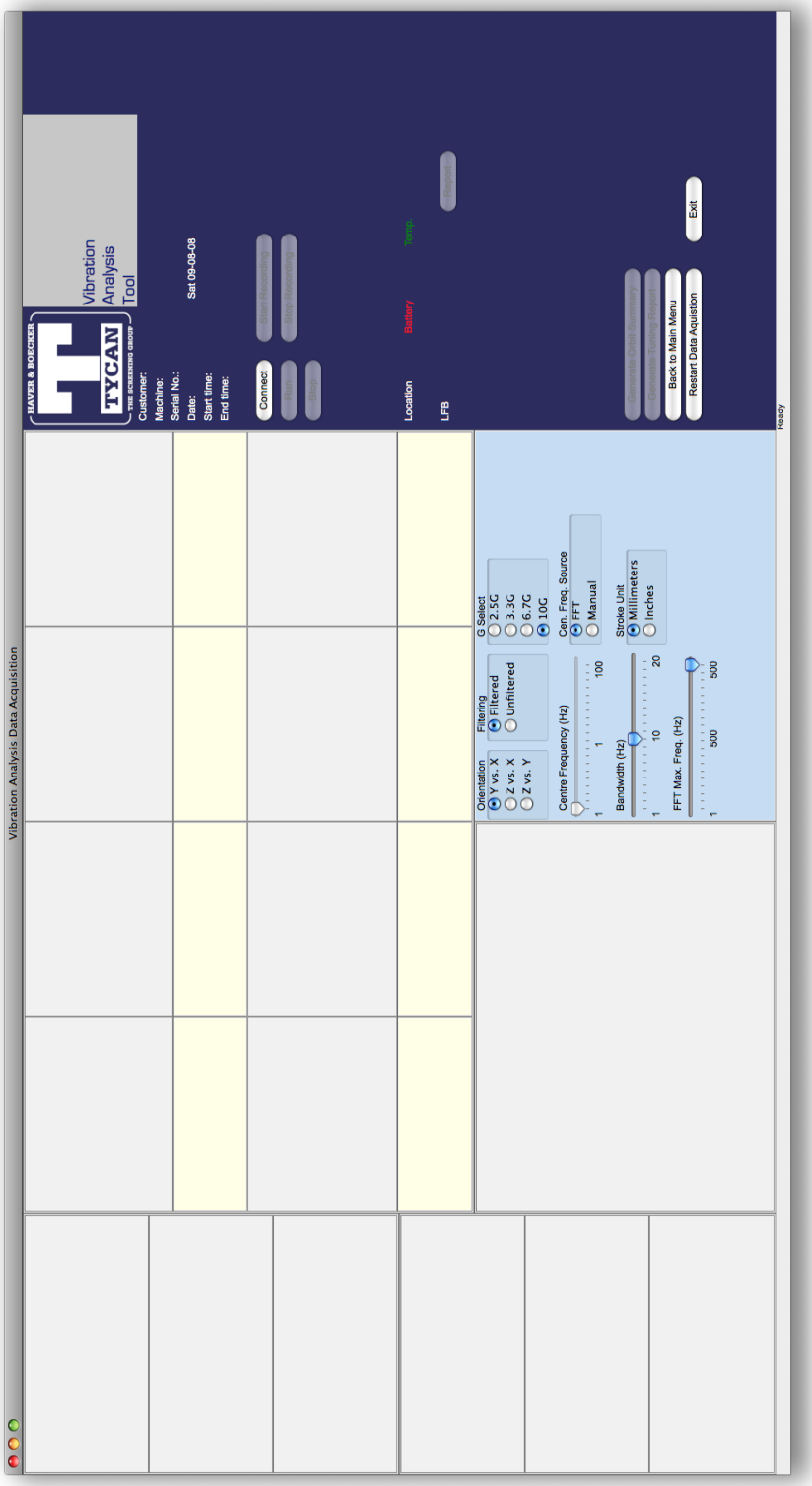


Figure A.7: Main view of Data Acquisition Mode.

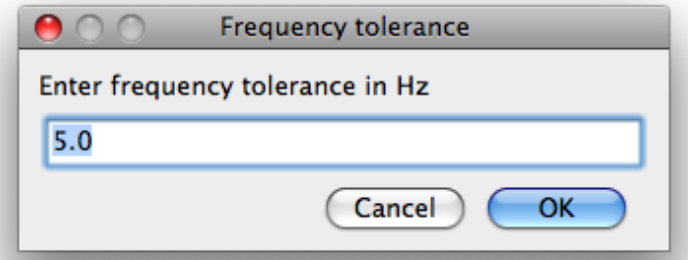


Figure A.8: Centre frequency tolerance setting.

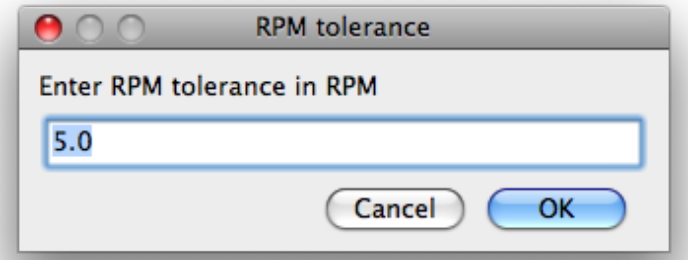


Figure A.9: RPM tolerance setting.

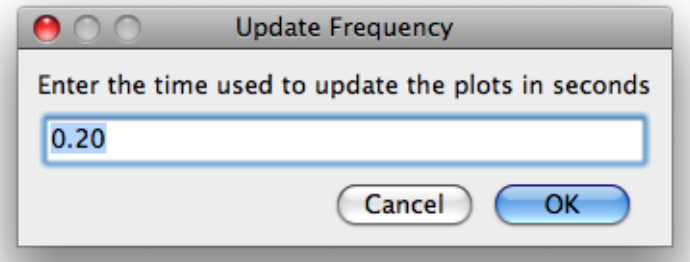


Figure A.10: Update frequency setting.

This parameter is measured in seconds (think of it as 5 seconds back in time for example), and can be changed as in Figure A.11.

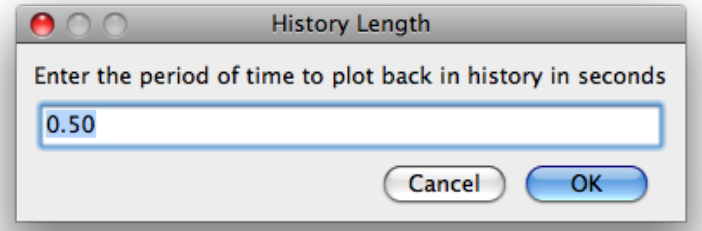


Figure A.11: Update history setting.

5. FFT Size: This parameter defines the size of data to perform the FFT computation on. The larger the size, the more accurate the results are, but the more expensive the computation becomes. This parameter has to be a power or two, and is therefore chosen from a list of predefined values as in Figure A.12.

6. Acceptance Levels: The acceptance levels correspond to the accepted deviations in the tuning report. These values are stored in a file and can be edited using this command. A window as in Figure A.13 will appear, and once all the values have been entered, the "Save" button should be pressed. If the user would like to go back to the original values prior to saving, the "Reset" button can be used. The "Exit" button will exit without saving.

7. Bearings: This command will allow you to add new bearing diameters along with their basic loads into a file where all the bearings are stored. This information is later used in the tuning report generation. When selecting this option, a window as in Figure A.14 will appear. You can add a new bearing by entering its diameter in millimetres in the "Bearing Diameter" box, and its basic dynamic load in kilo-Newton in the "Basic Dynamic Load" box. Press the "Add" button to add the requested bearing information. If a bearing with

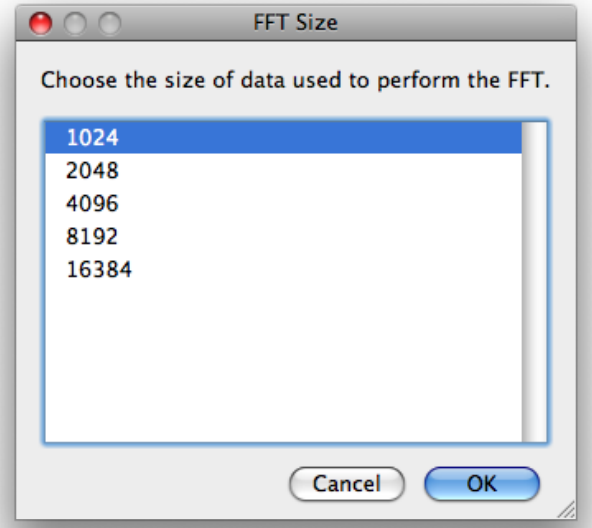


Figure A.12: FFT size setting.

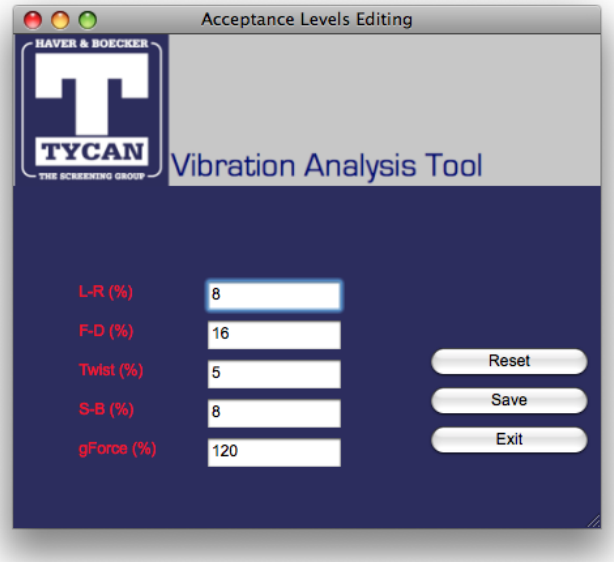


Figure A.13: Acceptance levels setting.

the inputted diameter already exists, a message will appear explaining that. If the input was accepted, the bearing will be added to the "Available Bearing Diameters and their Basicloads" list. Once you are done, press the "Exit" button.

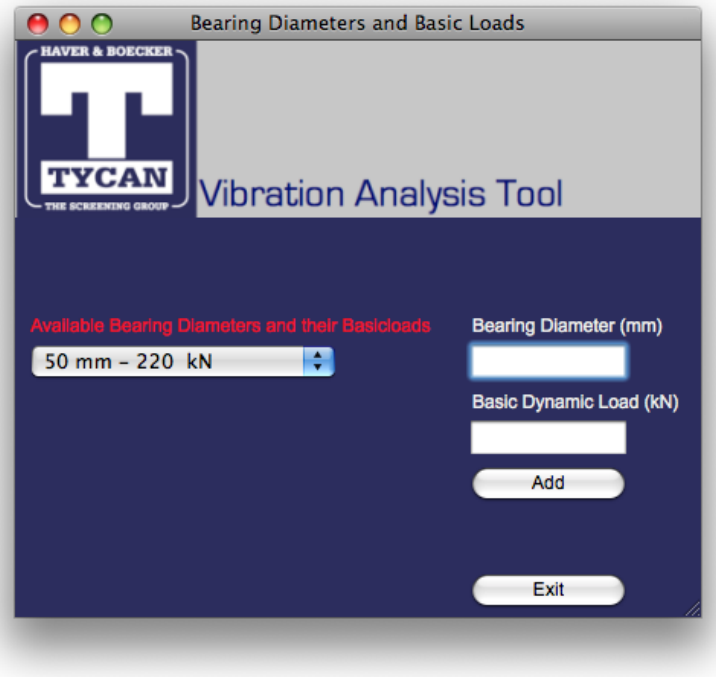


Figure A.14: Bearings settings.

A.5 Generating Reports

In order to generate reports in "Data Acquisition" mode, a recording of data must have taken place. On the other hand, in "File Simulation" mode, reports can be generated right away after connecting to the files.

Single point reports can be generated by clicking on the "Report" button beside each appropriate device on the right side of the application. An orbit summary report can be generated by clicking on the "Generate Orbit Summary" button, and a tuning

report can be generated by clicking on the "Generating Tuning Report" button. The latter will lead to a window as displayed in Figure A.15. Here, the user can insert the parameters that are to be used for generating the tuning report. The user can press the "Preview Results" button to preview the results that would show up in the report if they were to generate it. This gives the experienced user the chance to adjust the parameters in order to come up with the best results. Additionally, the g values are displayed in this window as computed from the data, and can be adjusted by the user. For a 2 bearing screen, 4 locations will appear, and for a 4 bearing there will be 8 locations.

A.5.1 Other Commands

The options panel includes the following controllers which the user can change anytime while the data is being processed:

- A radio-box for selecting the orientation to be displayed on the large plot.
- A radio-box for selecting whether to display filtered or unfiltered data on the large plot.
- A radio-box for selecting the source of the centre frequency (from the FFT computation, or manually using the slider).
- A radio-box for selecting the unit the stroke is to be displayed in (mm or inches).
- A radio-box for selecting the g value the sensors should operate at. This is only enabled when the data reading is stopped and in data acquisition mode only.
- A slider for setting the centre frequency manually.
- A slider for setting the maximum frequency to be displayed on the x axis of the FFT plots. This acts as zooming in on the x axis.
- A slider for setting the bandwidth for the bandpass filter.

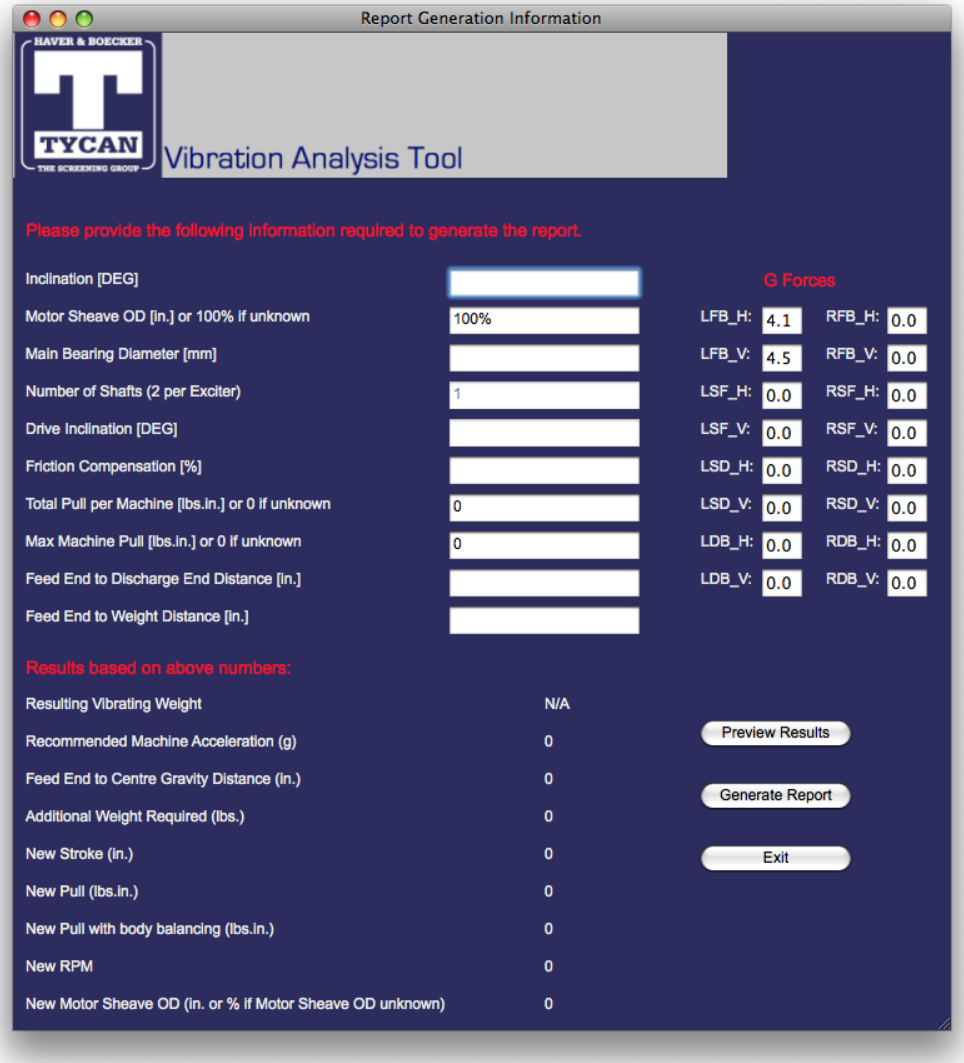


Figure A.15: Tuning Report Menu.

Additionally, the user can perform the following:

1. Restart the data acquisition by using the “Restart Data Acquisition” button which will take the user back to the acquisition information gathering window and will rediscover the available bluetooth devices.

2. Go back to the main menu by using the “Back to Main Menu” button, which will take the user back to the language selection menu.

3. Exit the application using the “Exit” button.

Appendix B

Internationalization

B.1 Editing the translation files

The following are the steps for editing the translation files:

1. Download and install a program called Poedit, which will make the task of applying the translation very easy. It is cross-platform and can be download it at: http://www.poedit.net/download.php

2. The folder 'locale' contains subfolders for each one of the used languages.
   de - German
   en - English (this does not require translation and should be left untouched)
   fr - French
   pt - Portuguese
   sp - Spanish

3. Inside each one of the language folders, there is a subfolder LC_MESSAGES which contains two files: gui_xx.mo and gui_xx.po where xx stands for the language abbreviations given above. Open the .po file using Poedit.

4. Once the .po file is opened, a list of the original strings in English used throughout the program is displayed. Upon clicking on a string, two boxes at the bot-

tom will show up. The top one contains the string in its original form, and the bottom one where appropriate translation should be entered. The translation should then show in the column entitled "Translation" beside the original string that was selected.

5. Once all the translations are applied, click "Save". This will overwrite the gui_xx.mo file with the new translations.

B.2 Adding a language

To add a new language, perform the following steps:

1. Add the language to the drop down list in the language selection frame.
2. Add the following line at the top of the appropriate viewer module:
   language = gettext.translation('gui_xx', './locale', languages = ['xx'])
   Where *language* is the language name and *xx* is the language abbreviation.
3. In the locale folder, create a folder and name it with the language's abbreviation.
4. Inside the new language folder, create an LC_MESSAGES folder.
5. Copy the 'messages.pot' file from the 'gui' folder and paste it in the new LC_MESSAGES folder.
6. Rename the file 'gui_xx.po' where xx is the language abbreviation for the new language being added.
7. Perform the steps required for editing the translation file as in Section B.1.

B.3 Adding new strings to the source code

If new translatable strings are added to the source code, perform the following to translate them:

1. Make sure they appear between _( ). For example: _("This is a translatable string").

2. Open the *app.fil* file under the gui folder and make sure that the python modules with the new translatable strings are added to the file.

3. On the command line, go to the gui folder and run:
   python mki18n.py -p
   This will create a new 'messages.pot' file with the new string(s) added to it.

4. For each language, open its .po file using Poedit, and then select 'Update from pot catalog" and choose 'messages.pot'. This will update the .po file with the new strings.

5. Perform the steps required for editing the translation files as in Section B.1 to apply the translation to the added string(s).

Appendix C

Reports

C.1 Single Point Report

Figure C.1 is an example of a single point report. The report includes the following:

1. An orbit plot with the filtered and unfiltered data orbits plotted on top of each-other to allow for analysing the data better.
2. Waveform plots for the x,y, and z axes.
3. FFT plots for the x,y, and z axes.
4. A table summarizing the acceleration, stroke, RPM and phase angle as acquired from that location on the machine.
5. A table summarizing the accelerations and strokes for each of the x,y, and z axes.

Date: Wed 16-07-08
Customer: test3
Machine:
No. Bearings:2
Serial No.:
Start Time: 15h15m15s
End Time: 15h18m02s

HAVER & BOECKER TYCAN THE SCREENING GROUP Vibration Analysis Tool

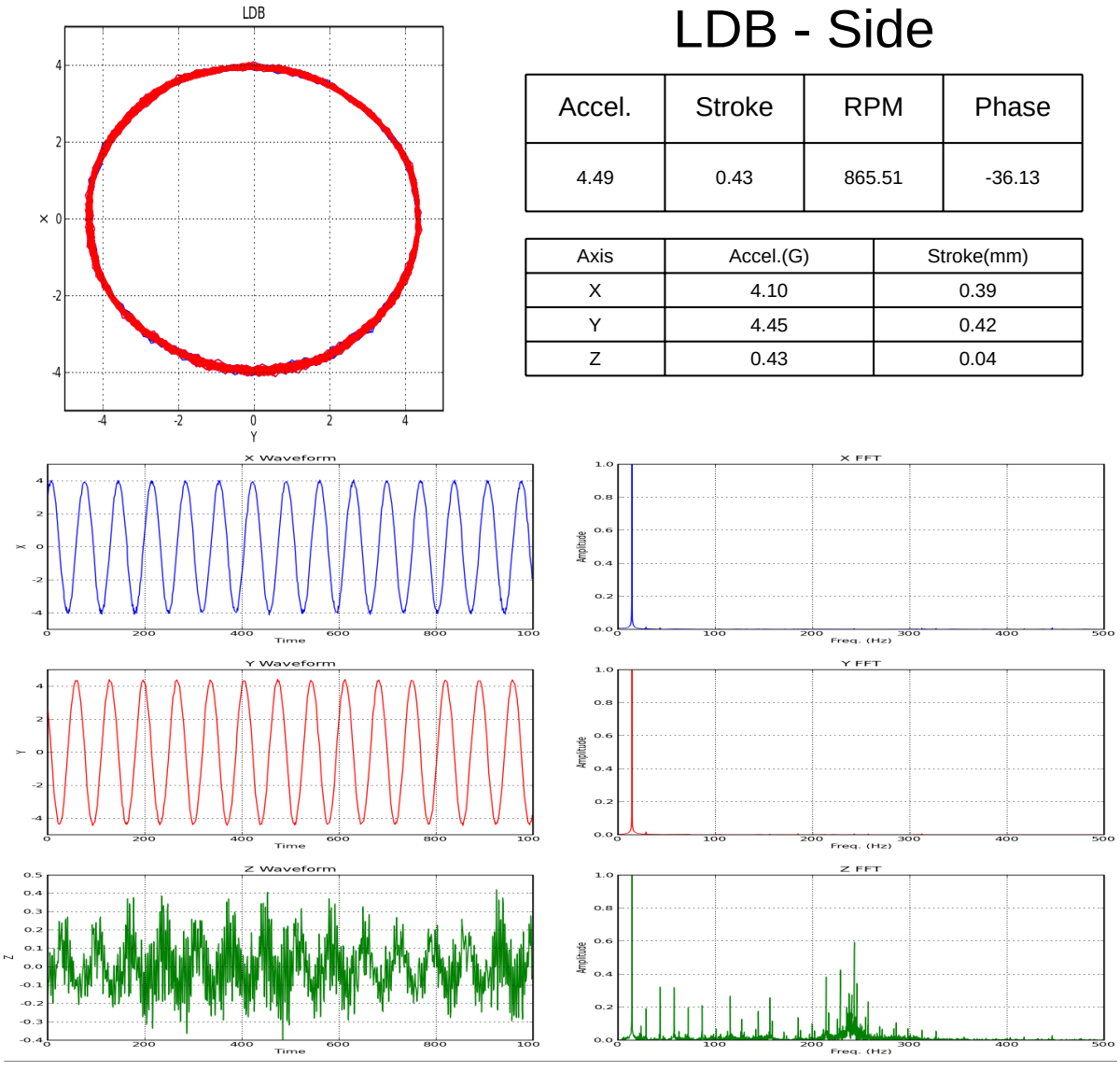


Figure C.1: Example of a single point report.

C.2 Orbit Summary Report

Figure C.2 is an example of an orbit summary report. The report includes the following:

1. For each of the monitored locations, an orbit plot representing the filtered data coming from the sensor on that location.

2. For each of the monitored locations, a table summarizing the main g force, the phase angle. and the x, y, and z g forces.

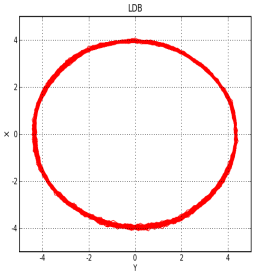
The orbit summary report allows for comparing the orbits visually as well as by the parameters provided in the tables.

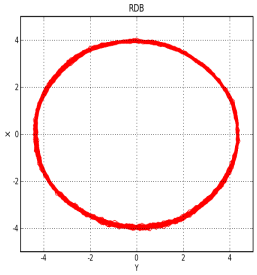
Date: Wed 16-07-08
Customer: test3
Machine:
No. Bearings:2
Serial No.:
Start Time: 15h15m15s
End Time: 15h18m02s

HAVER & BOECKER
TYCAN
THE SCREENING GROUP
Vibration Analysis Tool

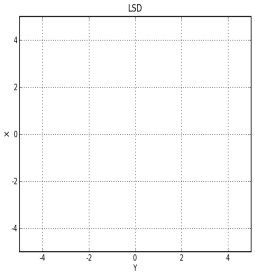
Discharge

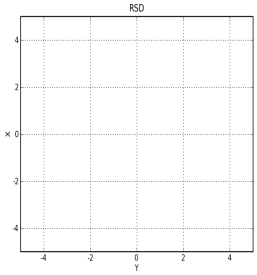
LDB	
Main G	4.5
Phase	-36.1
X	4.5
Y	4.1
Z	0.4

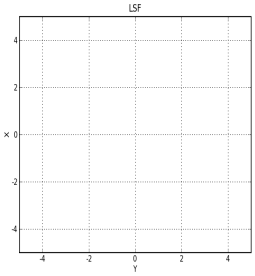


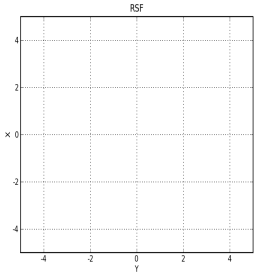


RDB	
Main G	4.5
Phase	-36.1
X	4.5
Y	4.1
Z	0.4

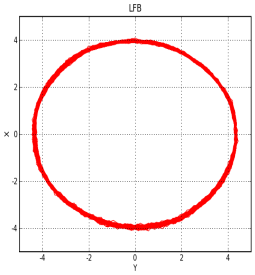




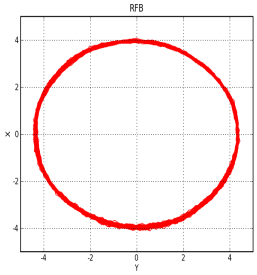




LFB	
Main G	4.5
Phase	-36.1
X	4.5
Y	4.1
Z	0.4



Feed



RFB	
Main G	4.5
Phase	-36.1
X	4.5
Y	4.1
Z	0.4

TYCAN Vibration Analysis

Figure C.2: Example of an orbit summary report.

C.3 Tuning Report

Figure C.3 is an example of a tuning report for a 2 bearing screen. The report includes the following information:

1. A table of deviations. The values appear in red if they exceed the acceptance levels shown in the "Deviation Acceptance Levels" table, which can be adjusted by the user through the "Settings" menu in the application.

2. A table summarizing the g forces and stroke values.

3. A table displaying the RPM measured, the inclination (entered by the user), the motor sheave (entered by the user), the number of shafts (entered by the user), the drive inclination (entered by the user), and the calculated recommended machine acceleration.

4. A table displaying the bearing diameter, the friction compensation, the total pull per machine and the maximum machine pull which are all entered by the user. Additionally, the resulting vibrating weight is calculated and displayed.

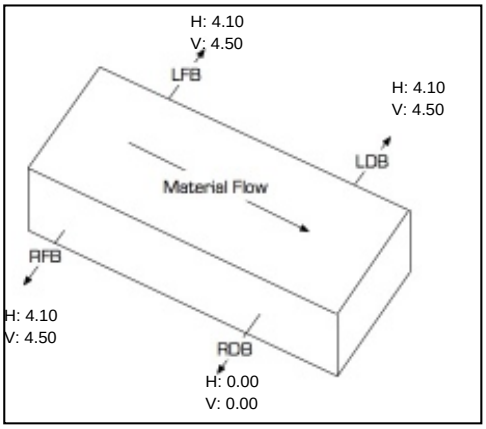
5. A 'Recommendations' table providing recommendations to change the stroke by changing the pull to a calculated value, with a body balance of a calculated value, or to change the speed to a calculated recommended RPM by changing the motor sheave OD to a calculated value.

6. A table providing estimations on the expected lifetime of the machine.

Date: Wed 16-07-08
Customer: test3
Machine:
No. Bearings:2
Serial No.:
Start Time: 15h15m15s
End Time: 15h18m02s

HAVER & BOECKER TYCAN THE SCREENING GROUP Vibration Analysis Tool

Deviation Acceptance Levels			
L-R : 8%	F-D : 16%	Twist : 5%	gForce : 120%



Deviations

Left - Right		Feed - Discharge		X Twist
DH	200.0%	LH	0.0%	Horizontal
DV	200.0%	LV	0.0%	133.3%
FH	0.0%	RH	200.0%	Vertical
FV	0.0%	RV	200.0%	133.3%

Legend

H	Horizontal	D	Discharge-End
V	Vertical		
F	Feed-End		

Summary	Horizontal	Vertical	Main
Max g-Force	4.10	4.50	4.30
Ave. g-Force	3.07	3.38	3.22
Max Stroke (in.)	0.39	0.43	0.41
Ave. Stroke (in.)	0.29	0.32	0.31

Bearing Diameter	70.0 mm
Friction Compensation	40.0%
Total Pull per Machine (lbs.in.)	90.0
Max. Machine Pull (lbs.in.)	0.0
Resulting Vibrating Weight	589.5 lbs

RPM measured	865.5
Inclination	20.0 DEG
Motor Sheave OD (in.)	100.0
No. Shafts (2 per Exciter)	1
Drive Inclination	30.0 DEG
Recommended Machine Acceleration	5.2 g

Recommendations

Body Balancing			Change Stroke to 0.5 in.	Or Change Speed To
Lengths	FB to DB	20.0 in.	By Changing Pull to:	1095.7 RPM
	FB to Weight	30.0 in.	144.2 lbs.in.	By Changing Motor
Results	FB to CG	0.0 in.	With Body Balance	Sheave OD to:
	Weight	294.8	216.3 lbs.in.	126.6 %

Estimations

L10	Existing Setup	New Stroke	New Speed	Stroke + Body Balance
Expl.	0h	0h	0h	0h

Clarify any adjustments with the manufacturer of the vibrating screen.

TYCAN Vibration Analysis

Figure C.3: Example of a tuning report.